HPC 3 Reduction code

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <climits>
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)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] > 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(vector<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;
arr.push_back(5);
arr.push_back(2);
arr.push_back(9);
arr.push_back(1);
arr.push_back(7);
arr.push_back(6);
arr.push_back(8);
arr.push_back(3);
arr.push_back(4);
min_reduction(arr);
max_reduction(arr);
sum_reduction(arr);
average_reduction(arr);
}
```

Explanation:

1. In the main function, a vector arr is declared using std::vector<int>. This vector will hold a collection of integer values.
2. The push_back method is used to add individual elements to the vector arr. In this case, the values 5, 2, 9, 1, 7, 6, 8, 3, and 4 are added to the vector.
3. The min_reduction function is called, passing the vector arr as an argument. This function will find and print the minimum value in arr.
4. The max_reduction function is called, passing the vector arr as an argument. This function will find and print the maximum value in arr.
5. The sum_reduction function is called, passing the vector arr as an argument. This function will calculate and print the sum of all the elements in arr.
6. The average_reduction function is called, passing the vector arr as an argument. This function will calculate and print the average value of the elements in arr.

Now, let's revisit each of the four reduction functions and explain them in more detail:

1. The min_reduction function:
- It takes a reference to a vector of integers arr as a parameter.
- It initializes the variable min_value to the maximum possible integer value using INT_MAX from climits.
- The #pragma omp parallel for reduction(min: min_value) directive is used to parallelize the subsequent loop and specify that a reduction operation should be performed to find the minimum value.
- The loop iterates over the elements of arr. Each thread compares the value at index i with the current minimum value (min_value) and updates min_value if a smaller value is found.
- After the parallel loop, the minimum value is printed to the console.

2. The max_reduction function:
- It is similar to min_reduction but finds the maximum value.
- The max_value variable is initialized to the minimum possible integer value using INT_MIN from climits.
- The #pragma omp parallel for reduction(max: max_value) directive specifies parallel execution and the reduction operation to find the maximum value.
- Inside the loop, each thread compares the value at index i with the current maximum value (max_value) and updates max_value if a larger value is found.
- After the parallel loop, the maximum value is printed to the console.

3. The sum_reduction function:
- It calculates the sum of all elements in the vector arr.
- The sum variable is initialized to 0.
- The #pragma omp parallel for reduction(+: sum) directive enables parallel execution and specifies the reduction operation of adding each element to sum.
- Inside the loop, each thread adds the value at index i to the sum variable.
- After the parallel loop, the total sum is printed to the console.

4. The average_reduction function:
- It calculates the average of the elements in the vector arr.

• The sum variable is initialized to 0.
• The #pragma omp parallel for reduction(+: sum) directive enables parallel execution and specifies the reduction operation of adding each element to sum.
• Inside the loop, each thread adds the value at index i to the sum variable.
• After the parallel loop, the sum is divided by the size of the vector arr to calculate the average, which is then printed to the console.
In summary, the main function initializes a vector with integer values, and then the code calls different reduction functions to find the minimum and maximum values, calculate the sum, and compute the average of the elements in the vector. The OpenMP directives enable parallel execution of the corresponding loops, distributing the work among multiple threads

Oral question

Parallel Reduction Reduce is a collective communication primitive used in the context of a parallel programming model to combine multiple vectors into one, using an associative binary operator. Every vector is present at a distinct processor in the beginning. The goal of the primitive is to apply the operator in the order given by the process or induces to the vectors until only one is left.

Rest is given in the explanation itself.