HPC 2 BUBBLE SORT #include <iostream>

```
#include <iostream>
#include <vector>
#include <random>
#include <omp.h>
#include <chrono>

using namespace std;

void bubbleSort(vector<int>& arr) {
  int n = arr.size();

  #pragma omp parallel for
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

int main() {
  int n = 1000;

  vector<int> arr(n);

  // Generate random numbers
  random_device rd;
  mt19937 generator(rd());
  uniform_int_distribution<int> distribution(1, 1000);

  for (int i = 0; i < n; i++) {
    arr[i] = distribution(generator);
  }

  cout << "Unsorted array: ";
  for (int i = 0; i < n; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;

  // Bubble sort.
  cout << "Sequential bubble sort: ";
  auto start = chrono::high_resolution_clock::now();
  bubbleSort(arr);
  auto end = chrono::high_resolution_clock::now();
  chrono::duration<double> elapsed = end - start;
  cout << elapsed.count() << " seconds" << endl;

  cout << "Sorted array: ";
  for (int i = 0; i < n; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;

  // Parallel bubble sort.
  cout << "Parallel bubble sort: ";
  start = chrono::high_resolution_clock::now();
  #pragma omp parallel
```

```
 bubbleSort(arr);
 end = chrono::high_resolution_clock::now();
 elapsed = end - start;
 cout << elapsed.count() << " seconds" << endl;

 return 0;
}
```

Explanation of code:

The code demonstrates the implementation of sequential and parallel bubble sort algorithms using OpenMP. Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm continues until the entire list is sorted. The code uses 1000 sequentially randomly generated numbers

Here's a breakdown of the code:

1. The code starts by including the necessary header files for input/output, vector manipulation, random number generation, OpenMP, and time measurement.

2. The `bubbleSort` function takes a reference to a vector `arr` as a parameter and performs the bubble sort algorithm on the vector.

3. Inside the `bubbleSort` function, a parallel region is defined using `#pragma omp parallel for`. This directive indicates that the loop following it can be executed in parallel by multiple threads. Each thread will be responsible for executing a portion of the loop iterations.

4. The outer loop runs from 0 to `n-1` (where `n` is the size of the vector) and represents the number of passes needed to sort the vector.

5. The inner loop runs from 0 to `n-i-1` and performs the comparisons and swaps between adjacent elements.

6. If the current element is greater than the next element, a swap is performed using a temporary variable `temp`.

7. The main function begins by defining the size of the vector `n` and creating a vector `arr` with `n` elements.

8. Random numbers are generated using the `random_device`, `mt19937`, and `uniform_int_distribution` classes to populate the vector with values ranging from 1 to 1000.

9. The unsorted vector is printed to the console.

10. Sequential bubble sort is performed on the vector by calling the `bubbleSort` function.

11. The time taken to execute the sequential bubble sort is measured using the `chrono` library.

12. The sorted vector is printed to the console.

13. Parallel bubble sort is performed on the vector by using `#pragma omp parallel` before calling the `bubbleSort` function.

14. The time taken to execute the parallel bubble sort is measured using the `chrono` library.

15. The execution times for sequential and parallel bubble sort are printed to the console.

16. The program terminates, and the execution completes.

Overall, the code demonstrates the implementation of bubble sort and showcases how it can be parallelized using OpenMP to potentially improve the sorting performance by utilizing multiple threads.

ORAL QUESTIONS

1. What is Bubble Sort - It is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. It is called "bubble" sort because the algorithm moves the larger elements towards the end of the array in a manner that resembles the rising of bubbles.

2. Advantage and disadvantage - The time complexity of Bubble Sort is O(n^2), which makes it inefficient. The advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small dataset.

3. OpenMP ? - OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing cluster.

4. Parallel Bubble Sort - Parallel Bubble Sort is a modification of the classic Bubble Sort algorithm that takes advantage of parallel processing to speed up the sorting process. In parallel Bubble Sort, the list of elements is divided into multiple sublists that are sorted concurrently by multiple threads. Each thread sorts its sublist using the regular Bubble Sort algorithm. When all sublists have been sorted, they are merged together to form the final sorted list.

5. How to check CPU utilization and memory consumption in ubuntu -
   In Ubuntu, you can use a variety of tools to check CPU utilization and memory consumption. Here are some common tools:
   1) top: The top command provides a real-time view of system resource usage, including CPU
   2) utilization and memory consumption. To use it, open a terminal window and type
   3) htop: htop is a more advanced version of top that provides additional features, such as interactive
   4) process filtering and a color-coded display. To use it, open a terminal window and type htop.
   5) ps: The ps command provides a snapshot of system resource usage at a particular moment in time. To use it, open a terminal window and type ps aux. This will display a list of all running processes and their resource usage.
   6) free: The free command provides information about system memory usage, including total, used, and free memory. To use it, open a terminal window and type free -h.
   7) vmstat: The vmstat command provides a variety of system statistics, including CPU utilization, memory usage, and disk activity. To use it, open a terminal window and type vmstat