Parallel Bubble Sort

Title of the Assignment: Write a program to implement Parallel Bubble Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objective of the Assignment: Students should be able to Write a program to implement Parallel Bubble Sort and can measure the performance of sequential and parallel algorithms.

Prerequisite:

1. Basic of programming language
2. Concept of Bubble Sort
3. Concept of Parallelism

Contents for Theory:

1. **What is Bubble Sort? Use of Bubble Sort**
2. **Example of Bubble sort?**
3. **Concept of OpenMP**
4. **How Parallel Bubble Sort Work**
5. **How to measure the performance of sequential and parallel algorithms?**

What is Bubble Sort?

Bubble Sort 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 in a liquid.

The basic algorithm of Bubble Sort is as follows:

1. Start at the beginning of the array.
2. Compare the first two elements. If the first element is greater than the second element, swap them.
3. Move to the next pair of elements and repeat step 2.
4. Continue the process until the end of the array is reached.
5. If any swaps were made in step 2-4, repeat the process from step 1.

The time complexity of Bubble Sort is O(n^2), which makes it inefficient for large lists. However, it has the advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small datasets.

Bubble Sort has limited practical use in modern software development due to its inefficient time complexity of O(n^2) which makes it unsuitable for sorting large datasets. However, Bubble Sort has some advantages and use cases that make it a valuable algorithm to understand, such as:

1. Simplicity: Bubble Sort is one of the simplest sorting algorithms, and it is easy to understand and implement. It can be used to introduce the concept of sorting to beginners and as a basis for more complex sorting algorithms.
2. Educational purposes: Bubble Sort is often used in academic settings to teach the principles of sorting algorithms and to help students understand how algorithms work.
3. Small datasets: For very small datasets, Bubble Sort can be an efficient sorting algorithm, as its overhead is relatively low.
4. Partially sorted datasets: If a dataset is already partially sorted, Bubble Sort can be very efficient. Since Bubble Sort only swaps adjacent elements that are in the wrong order, it has a low number of operations for a partially sorted dataset.
5. Performance optimization: Although Bubble Sort itself is not suitable for sorting large datasets, some of its techniques can be used in combination with other sorting algorithms to optimize their performance. For example, Bubble Sort can be used to optimize the performance of Insertion Sort by reducing the number of comparisons needed.

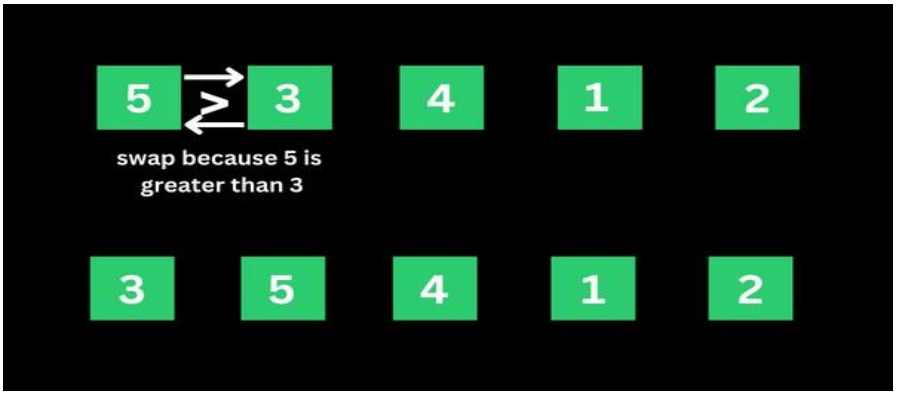
Example of Bubble sort

Let's say we want to sort a series of numbers 5, 3, 4, 1, and 2 so that they are arranged in ascending order…

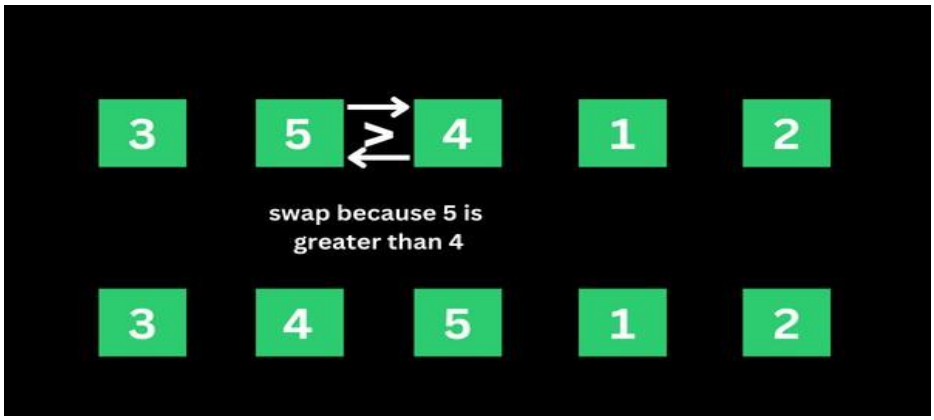
The sorting begins the first iteration by comparing the first two values. If the first value is greater than the second, the algorithm pushes the first value to the index of the second value.

First Iteration of the Sorting

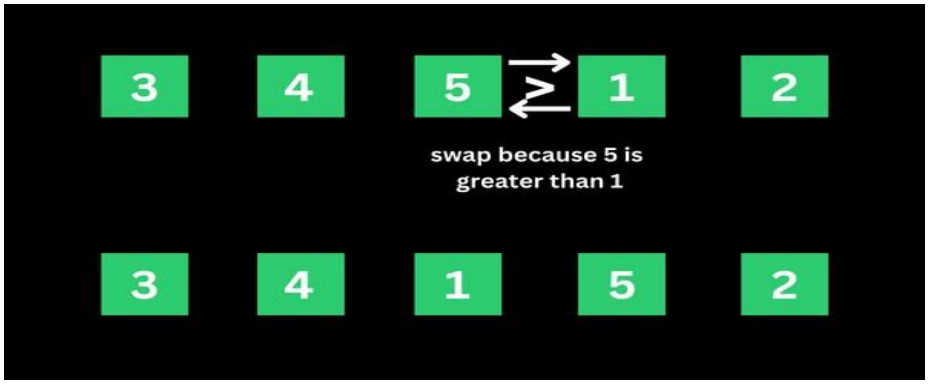
Step 1: In the case of 5, 3, 4, 1, and 2, 5 is greater than 3. So 5 takes the position of 3 and the numbers become 3, 5, 4, 1, and 2.



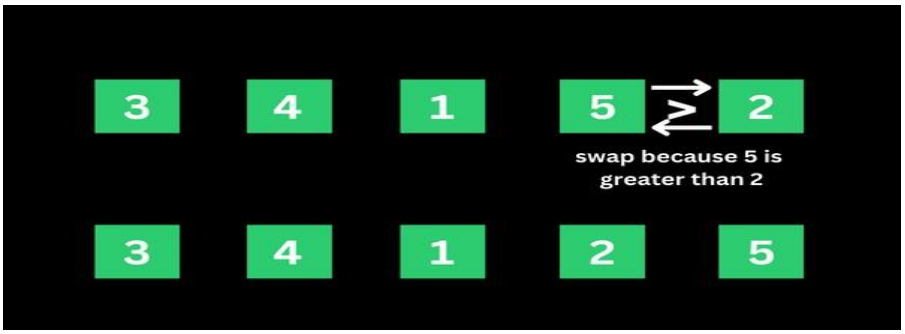
Step 2: The algorithm now has 3, 5, 4, 1, and 2 to compare, this time around, it compares the next two values, which are 5 and 4. 5 is greater than 4, so 5 takes the index of 4 and the values now become 3, 4, 5, 1, and 2.



Step 3: The algorithm now has 3, 4, 5, 1, and 2 to compare. It compares the next two values, which are 5 and 1. 5 is greater than 1, so 5 takes the index of 1 and the numbers become 3, 4, 1, 5, and 2.



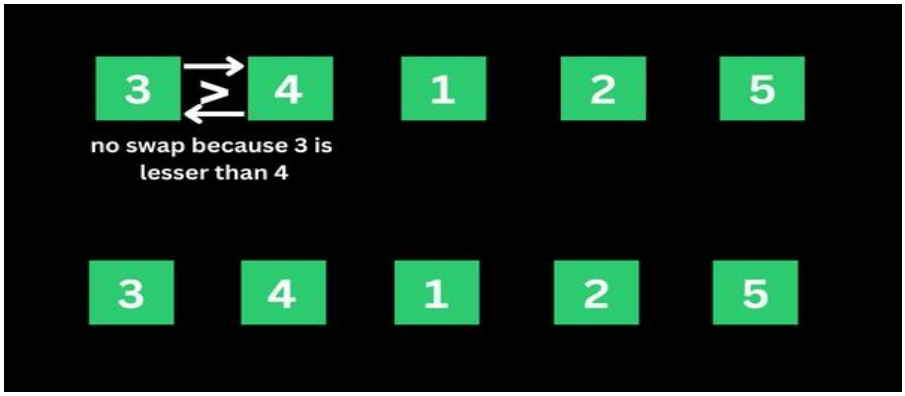
Step 4: The algorithm now has 3, 4, 1, 5, and 2 to compare. It compares the next two values, which are 5 and 2. 5 is greater than 2, so 5 takes the index of 2 and the numbers become 3, 4, 1, 2, and 5.



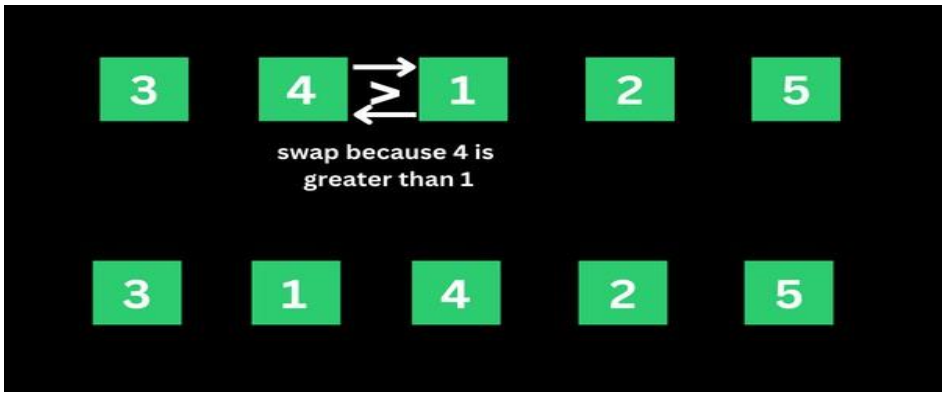
That's the first iteration. And the numbers are now arranged as 3, 4, 1, 2, and 5 – from the initial 5, 3, 4, 1, and 2. As you might realize, 5 should be the last number if the numbers are sorted in ascending order. This means the first iteration is really completed.

Second Iteration of the Sorting and the Rest

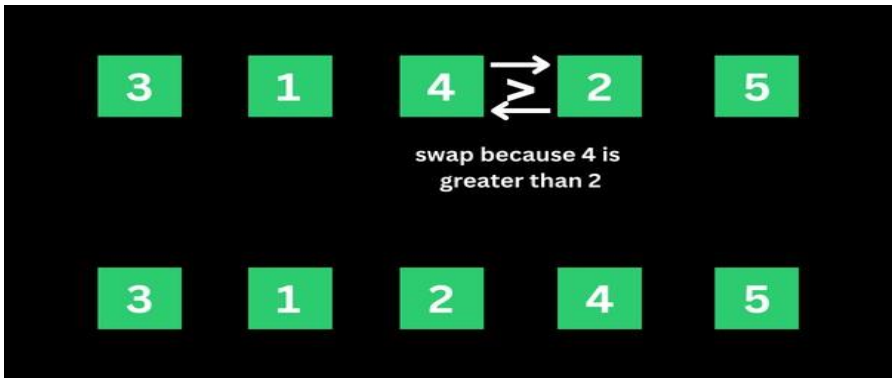
The algorithm starts the second iteration with the last result of 3, 4, 1, 2, and 5. This time around, 3 is smaller than 4, so no swapping happens. This means the numbers will remain the same.



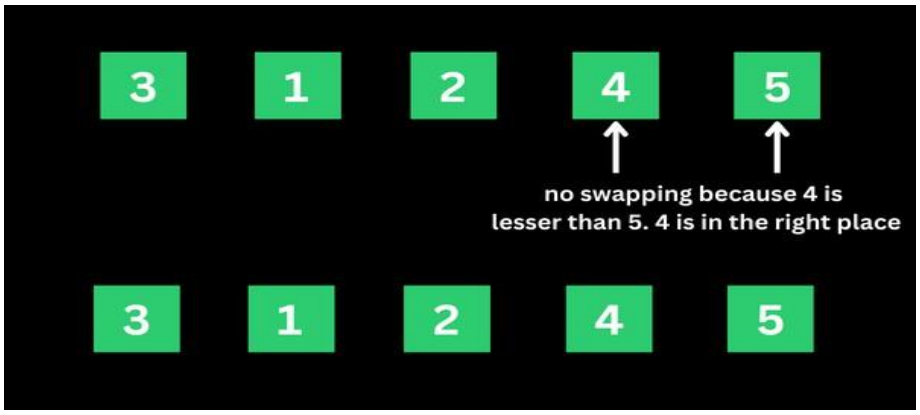
The algorithm proceeds to compare 4 and 1. 4 is greater than 1, so 4 is swapped for 1 and the numbers become 3, 1, 4, 2, and 5.



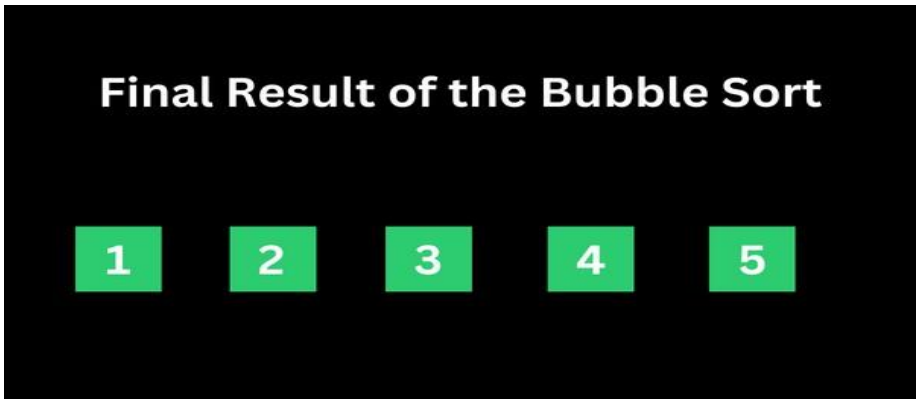
The algorithm now proceeds to compare 4 and 2. 4 is greater than 2, so 4 is swapped for 2 and the numbers become 3, 1, 2, 4, and 5.



4 is now in the right place, so no swapping occurs between 4 and 5 because 4 is smaller than 5.



That’s how the algorithm continues to compare the numbers until they are arranged in ascending order of 1, 2, 3, 4, and 5.



Concept of 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 clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel Bubble Sort Work

- 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.

- The parallelization of the algorithm is achieved using OpenMP, a programming API that supports parallel processing in C++, Fortran, and other programming languages. OpenMP provides a set of compiler directives that allow developers to specify which parts of the code can be executed in

parallel.

- In the parallel Bubble Sort algorithm, the main loop that iterates over the list of elements is divided into multiple iterations that are executed concurrently by multiple threads. Each thread sorts a subset of the list, and the threads synchronize their work at the end of each iteration to ensure that the elements are properly ordered.

- Parallel Bubble Sort can provide a significant speedup over the regular Bubble Sort algorithm, especially when sorting large datasets on multi-core processors. However, the speedup is limited by the overhead of thread creation and synchronization, and it may not be worth the effort for small datasets or when using a single-core processor.

How to measure the performance of sequential and parallel algorithms?

To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms.
3. Use a reliable timer to measure the execution time of each algorithm on each test case.
4. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

- Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

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 utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.
2. **htop**: htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.
3. **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.
4. **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.
5. **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.

Conclusion- In this way we can implement Bubble Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

Assignment Question

1. **What is parallel Bubble Sort?**
2. **How does Parallel Bubble Sort work?**
3. **How do you implement Parallel Bubble Sort using OpenMP?**
4. **What are the advantages of Parallel Bubble Sort?**
5. **Difference between serial bubble sort and parallel bubble sort**

Reference link

- https://www.freecodecamp.org/news/bubble-sort-algorithm-in-java-cpp-python-with-example-code/

Here, we're using the #pragma omp parallel directive to parallelize the code using OpenMP. Within the parallel block, we have a while loop that runs until the array is sorted. Inside the while loop, we have a #pragma omp for directive that parallelizes the for loop that performs the bubble sort.

Note that the flag variable is used to keep track of whether any swaps were made during an iteration of the for loop. If no swaps were made, then the array is already sorted and we can exit the loop.

explanation of each line of the code:

```c++
#include <iostream>
#include <omp.h>

using namespace std;
```

These lines are the standard C++ header files used for input/output operations and OpenMP, respectively.

```c++
void parallel_bubble_sort(int arr[], int n) {
    #pragma omp parallel
    {
        int i, temp, flag = 1;
        while (flag) {
            flag = 0;
            #pragma omp for
            for (i = 0; i < n-1; i++) {
                if (arr[i] > arr[i+1]) {
                    temp = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = temp;
                    flag = 1;
                }
            }
        }
    }
}
```

This is the function parallel_bubble_sort() that implements the parallel bubble sort algorithm using OpenMP.

- #pragma omp parallel is a directive that creates a team of threads to execute the parallel code inside the block. In this case, the block contains the code for bubble sort algorithm.
- int i, temp, flag = 1; declares the variables i, temp, and flag that will be used inside the while loop.
- while (flag) is a loop that runs until the flag variable is 0.
- flag = 0; sets the flag variable to 0 before starting each iteration of the for loop.
- #pragma omp for is a directive that parallelizes the for loop, by dividing the loop iterations among the threads in the team. Each thread performs the sorting operation on a subset of the array, thereby making the sorting process faster.
- for (i = 0; i < n-1; i++) is a for loop that iterates over the array, from 0 to n-1.
- if (arr[i] > arr[i+1]) checks if the current element is greater than the next element.
- temp = arr[i]; arr[i] = arr[i+1]; arr[i+1] = temp; swaps the current element with the next element, using a temporary variable.

- flag = 1; sets the flag variable to 1, indicating that a swap has been made.
- Finally, the sorted array is printed using a for loop.

```c++
int main() {
    int arr[] = {5, 3, 1, 9, 8, 2, 4, 7, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    parallel_bubble_sort(arr, n);

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

    return 0;
}
```

This is the main() function, which initializes an array arr and its size n. The function parallel_bubble_sort() is called with these arguments to sort the array. The sorted array is then printed to the console using a for loop.

How to Run code in Ubuntu:

1. **Open a terminal window.**

Compile the code using the following command:

```
g++ -fopenmp filename.cpp -o executable
```

Run the program by executing the following command:

```bash
./executable
```

Output

```c
Sorted array: 1 2 3 4 5 6 7 8 9
```

This is because the input array `{5, 3, 1, 9, 8, 2, 4, 7, 6}` is sorted in ascending order using the parallel bubble sort algorithm implemented in the `parallel_bubble_sort()` function. The sorted array is then printed to the console in the `main()` function using a for loop.

To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms.
3. Use a reliable timer to measure the execution time of each algorithm on each test case.
4. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

- Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

How to check CPU utilisation 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 utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.
2. htop: htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.
3. 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.
4. 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.
5. 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.