**1.Include necessary headers:**

#include <iostream>

#include <omp.h>

#include <vector>

#include <cstdlib>

#include <ctime>

* iostream: Standard input/output library.
* omp.h: OpenMP library for parallel processing.
* vector: Dynamic array for storing elements.
* cstdlib and ctime: Used for generating random numbers.

**2. Swap function:**  
void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

* + Swaps two elements.

**3. Sequential Bubble Sort:**  
void bubbleSort(vector<int> &arr) { ... }

* + Uses nested loops to repeatedly swap adjacent elements.

**4. Parallel Bubble Sort (Using OpenMP):**  
#pragma omp parallel for

* + The #pragma omp parallel for directive allows multiple threads to process different iterations of the inner loop concurrently.

**5. Main function:**  
int main() { ... }

* + Generates an array of random numbers.
  + Measures execution time for both sequential and parallel Bubble Sort.
  + Uses omp\_get\_wtime() to measure time.

## **Compiling and Running the Program**

### **Compile:**

Use the following command to compile the program with OpenMP support:

g++ -fopenmp parallel\_bubble\_sort.cpp -o parallel\_bubble\_sort

### **Run:**

./parallel\_bubble\_sort

#include <iostream>

#include <omp.h>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

// Swap function to swap two elements

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

// Sequential Bubble Sort

void bubbleSort(vector<int> &arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

// Parallel Bubble Sort using OpenMP

void parallelBubbleSort(vector<int> &arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

// Parallel for loop with OpenMP

#pragma omp parallel for

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

int main() {

const int size = 1000; // Size of the array

vector<int> arr1(size), arr2(size);

srand(time(0));

// Generate random numbers

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

arr1[i] = arr2[i] = rand() % 10000;

}

// Measure time for Sequential Bubble Sort

double start = omp\_get\_wtime();

bubbleSort(arr1);

double end = omp\_get\_wtime();

cout << "Sequential Bubble Sort Time: " << (end - start) << " seconds" << endl;

// Measure time for Parallel Bubble Sort

start = omp\_get\_wtime();

parallelBubbleSort(arr2);

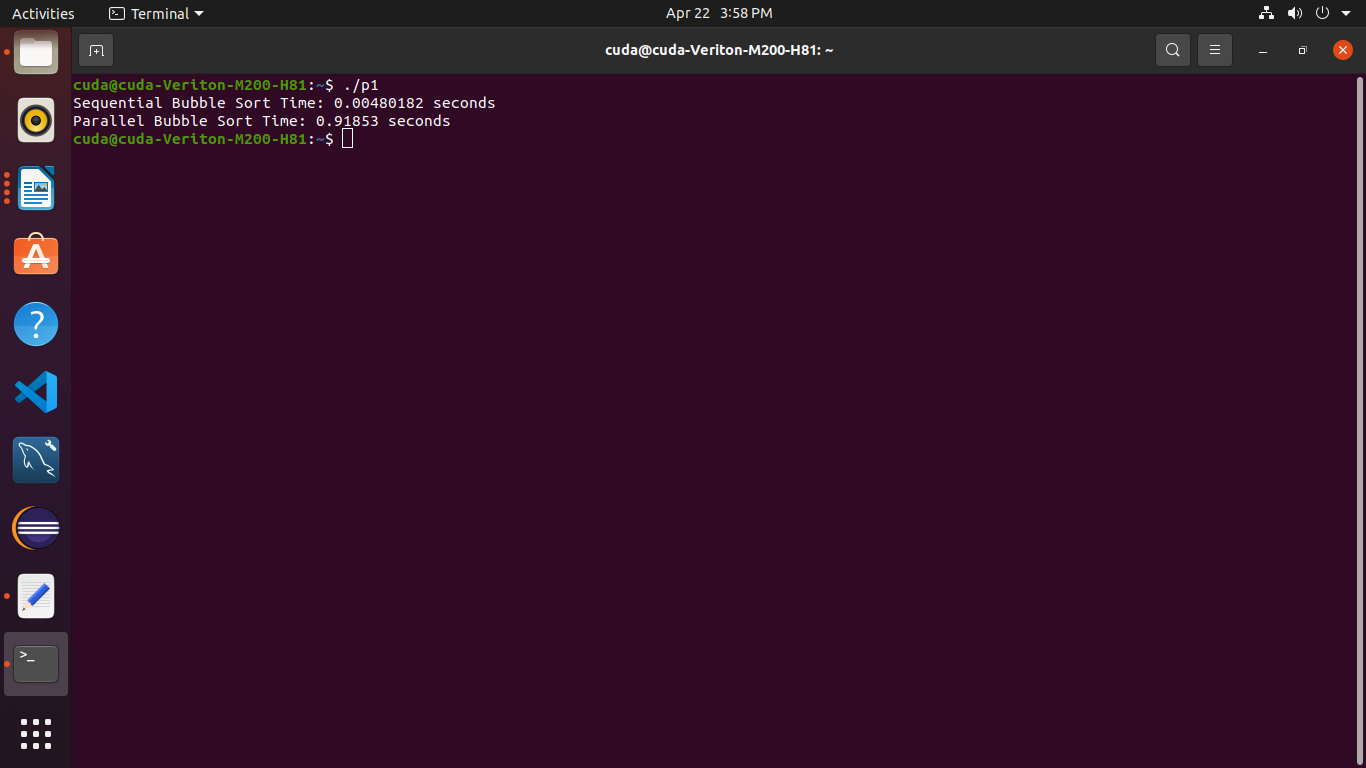
end = omp\_get\_wtime();

cout << "Parallel Bubble Sort Time: " << (end - start) << " seconds" << endl;

return 0;

}

**Output:**

****