Problem Statement: Implement HPC application for AI/ML domain.

Output

```
#include <iostream>
#include <vector>
#include <cmath>
#include <omp.h>
#include <algorithm>
using namespace std;

struct Neighbor {
    double distance;
    int label;
};
double euclidean_distance(const vector<double>& a, const vector<double>& b) {
    double sum = 0.0;
    for (size_t i = 0; i < a.size(); i++) {
        sum += (a[i] - b[i]) * (a[i] - b[i]);
    }
    return sqrt(sum);
}
int main() {
    int n, dims, k;
    cout << "Enter number of training samples: ";
    cin >> n;
    cout << "Enter number of features/dimensions: ";
    cin >> dims;
    vector<vector<double>> train_data(n, vector<double>(dims));
    vector<int> train_labels(n);
    cout << "Enter training data (space-separated features):\n";
    for (int i = 0; i < n; ++i) {
        cout << "Sample " << i + 1 << ": ";
        for (int j = 0; j < dims; ++j) {
            cin >> train_data[i][j];
        }
        cout << "Label for this sample: ";
        cin >> train_labels[i];
}

    vector<double> test_point(dims);
    cout << "Enter test point:\n";
    for (int j = 0; j < dims; ++j) {
        cout << "Feature " << j + 1 << ": ";
```

```
    cin >> test_point[j];
    }

    cout << "Enter value of K: ";
    cin >> k;
    vector<Neighbor> neighbors(n);
    #pragma omp parallel for
    for (int i = 0; i < n; ++i) {
        double dist = euclidean_distance(train_data[i], test_point);
        neighbors[i] = { dist, train_labels[i] };
    }

    sort(neighbors.begin(), neighbors.end(), [](Neighbor a, Neighbor b) {
        return a.distance < b.distance;
    });

    vector<int> count(100, 0); // Assuming labels are 0–99

    for (int i = 0; i < k; ++i) {
        count[neighbors[i].label]++;
    }

    int predicted_label = max_element(count.begin(), count.end()) - count.begin();
    cout << "\nPredicted label for the test point is: " << predicted_label << endl;

    return 0;
}
```