**Program 5**

#include <iostream>

#include <cstdlib> // or <stdlib.h> rand, srand

#include <ctime> // or <time.h> time

#include <omp.h>

#include <math.h>

#define K 4

using namespace std;

int num\_threads;

long num\_points;

long\*\* points;

// 2D array points[x][0] -> point location, points[x][1] -> distance from cluster mean

int cluster[K][2] = {

{75, 25}, {25, 25}, {25, 75}, {75, 75}

};

long cluster\_count[K];

void populate\_points() {

// Dynamically allocate points[num\_points][2] 2D array

points = new long\*[num\_points];

for (long i=0; i<num\_points; i++)

points[i] = new long[2];

// Fill random points (0 to 100)

srand(time(NULL));

for (long i=0; i<num\_points; i++) {

points[i][0] = rand() % 100;

points[i][1] = rand() % 100;

}

// Initialize cluster\_count

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

cluster\_count[i] = 0;

}

}

double get\_distance(int x1, int y1, int x2, int y2) {

int dx = x2-x1, dy = y2-y1;

return (double)sqrt(dx\*dx + dy\*dy);

}

void classify\_points() {

#pragma omp parallel for num\_threads(num\_threads)

for (long i=0; i<num\_points; i++) {

double min\_dist = 1000, cur\_dist = 1;

int cluster\_index = -1;

for (int j=0; j<K; j++) {

cur\_dist = get\_distance( points[i][0], points[i][1], cluster[j][0], cluster[j][1] );

if (cur\_dist<min\_dist)

{

min\_dist = cur\_dist;

cluster\_index = j;

}

}

cluster\_count[cluster\_index]++;

}

}

int main(int argc, char\* argv[])

{

num\_points = atol(argv[1]);

num\_threads = atoi(argv[2]);

populate\_points();

double t1 = omp\_get\_wtime();

classify\_points();

double t2 = omp\_get\_wtime();

double t = (t2 - t1) \* 1000;

cout<< "Time Taken: " << t << "ms" <<endl;

}