Program 5

#include &lt;iostream&gt;

#include &lt;cstdlib&gt; // or &lt;stdlib.h&gt; rand, srand

#include &lt;ctime&gt; // or &lt;time.h&gt; time

#include &lt;omp.h&gt;

#include &lt;math.h&gt;

#define K 4

using namespace std;

intnum\_threads;

longnum\_points;

long\*\* points; // 2D array points[x][0] -&gt; point location, points[x][1] -&gt; distance from

cluster mean

int cluster[K][2] = {

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

};

longcluster\_count[K];

voidpopulate\_points() {

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

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

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

points[i] = new long[2];

// Fill random points (0 to 100)

srand(time(NULL));

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

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

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

}

// Initialize cluster\_count

for (inti=0; i&lt;K; i++) {

cluster\_count[i] = 0;

}

}

doubleget\_distance(int x1, int y1, int x2, int y2) {

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

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

}

voidclassify\_points() {

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

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

doublemin\_dist = 1000, cur\_dist = 1;

intcluster\_index = -1;

for (int j=0; j&lt;K; j++) {

cur\_dist = get\_distance(

points[i][0], points[i][1],

cluster[j][0], cluster[j][1]

);

if (cur\_dist&lt;min\_dist) {

min\_dist = cur\_dist;

cluster\_index = j;

}

}

cluster\_count[cluster\_index]++;

}

}

int main(intargc, 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&lt;&lt; &quot;Time Taken: &quot; &lt;&lt; t &lt;&lt; &quot;ms&quot; &lt;&lt;endl;

}