REPORT

High Level Idea:

As we are trying to find the point, nearest(with minimum distance) to a given point, from a set of points, we divide the set into smaller subsets of approximately equal size equal to (number of points)/(number of threads). Each thread finds the point with minimum distance from a given subset. Now, as we get the points with minimum distances from each subset to the given point, among them we find the point corresponding to the least distance.

Description of Low-Level Design:

To explain the low-level design we could follow the control flow.

Initially the program takes the input from a file and stores the values of number of threads, coordinates of points, number of points. Now the program creates the threads according to the value of number of threads. It gives the min_dist() function as a thread function to each thread and assign each thread a subset of almost equal size (size of set/number of threads).

The min_dist() thread function identifies the start,end points of the subset assigned to it and it calls the find_min_dist() function.

The find_min_dist() function finds the point, which is at minimum distance from the given point, from the subset.
It stores the corresponding distance and the point in a global array.
(As we have to access this data from the main thread after).

After the completion of execution of the threads, the main thread then finds the corresponding point with minimum distance from the given point among the points returned by the threads.

Calculating distance:

If $\sqrt{(x1-x)^2 +(y1-y)^2} < \sqrt{(x2-x)^2 +(y2-y)^2}$
then $(x1-x)^2 +(y1-y)^2 < (x2-x)^2 +(y2-y)^2$

While calculating distance using the formula sqrt((x1-x2)² + (y1-y2)²), we ignore the sqrt() as we only need comparison between the distances and not the actual distances. By doing this we could avoid floating point accuracy problems while comparing.

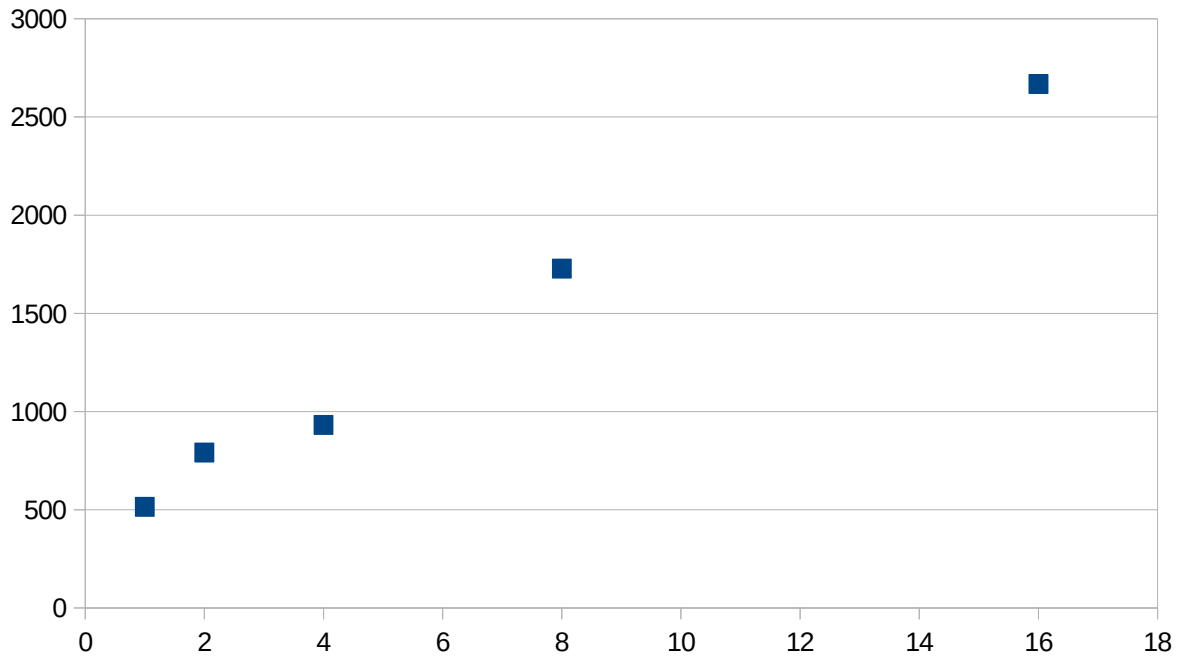
Measuring the time:

We read the clock just after taking input and just before printing output. The difference of these both values gives the total number of clocks required for execution. On dividing it with CLOCKS_PER_SEC we get the number of seconds taken for execution and on multiplying with 1000000 we get the execution time in microseconds.

COMPARISONS:

Graph plots:
1) Varying the number of threads.

(a) x-axis: number of threads varying as 1, 2, 4 and 8, 16.
(b) y-axis: time taken to find the point nearest to a given source from the given set of 5000 points.



2)Varying the input set size

(a) x-axis: the destination set sizes as 1000, 2000, 3000, 4000, 5000.
(b) y-axis: time taken to find the point nearest to the given source point while having the total number of threads as 16.