High Performance Networked Resources / Grid Computing

Assignment #1

MPI Program for finding the closest pair of a set of points

1. The problem

The problem in a two-dimensional space consists more precisely in the following: we are given a set **P** of **n** points in $\Re^2$ – where each point is represented by a pair of coordinates x = (x1, x2) – and we are interested in a pair of points **p**, **q** $\in$ **P** such that:

$|p - q| = \min\{|p' - q'| : p', q' \in \mathbf{P}, p' \neq q'\}$, where $|p - q| = \sqrt{((p1 - q1)^2 + (p2 - q2)^2)}$

is the Euclidean distance between p = (p1, p2) and q = (q1, q2).

The 1-dimensional version of this problem is equivalent to finding the smallest interval determined by **n** numbers, which is obviously determined by two consecutive numbers. To solve this problem, it suffices to sort the numbers and compute differences, and this results in a running time of O(n log n).

The brute force algorithm, which computes the distance between every point and all the others takes $O(n^2)$.

A divide-and-conquer algorithm exists for two and higher dimensions taking also O(n log n).

2. Divide and conquer

In two dimensions, a natural way to divide the problem is through a line in the domain space, particularly a vertical one. This divides the set **P** of points into $\mathbf{P_L}$ and $\mathbf{P_R}$ for which the problem is solved recursively. It remains to specify how to merge these sub-problems.

A general solution with divide and conquer, could be the recursive function

```
Closest-Pair-DC (P)
    1. Partition P with a vertical line L, halving the domain into points to the left $P_L$ and to the right $P_R$
    2. Let $(p_L, q_L)$ = Closest-Pair-DC($P_L$)
    3. Let $(p_R, q_R)$ = Closest-Pair-DC($P_R$)
    4. Let $\delta = \min\{ |p_L - q_L|, |p_R - q_R|\}$
    5. Let $(p_M, q_M)$ = CP-Merge(P, L, $\delta$)
    6. Return closest pair among $(p_L, q_L)$, $(p_R, q_R)$ and $(p_M, q_M)$
```

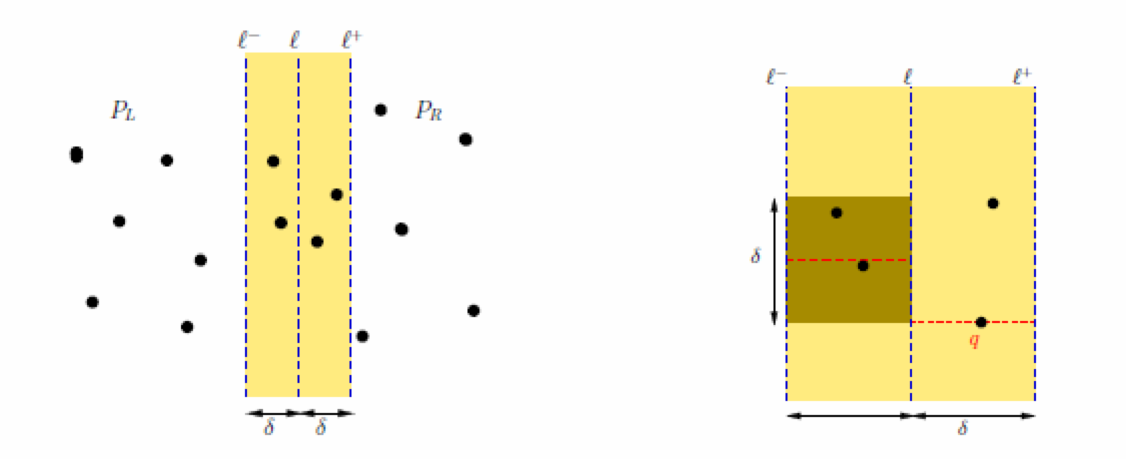
From the recursion, we know the closest pairs with both points on the left and on the right of L. It remains to check if there is a closest pair with one point on the left and one on the right of L. In principle, we would have to check every such pair, and that would be very time consuming.

We observe that:

(i) Only points within a strip bounded by lines L^- and L^+ parallel to L and at distance δ on the left and right (see next figure on the left) are of interest: a point outside this strip would be more than a distance δ away from a point on the opposite side. Let $\mathbf{P_{strip}}$ denote the subset of **P** in this strip (±δ from L).

(ii) Each point in this strip only need to be checked against four other points, that can be easily determined. To see this, consider the lowest point **q** of a possible closest pair. Suppose **q** is on the right. Then a candidate **p** to be closest to **q** must lie in the square determined by the lines L-δ, L, x2 = q2 and x2 = q2 + δ (see right figure). This square contains at most 4 points: if there were more than 4, then there should be 2 in one of the four "sub-squares" of this one with side δ/2 (see figure) and so the distance between them is at most δ/√2, which is less than δ (which is impossible, being δ the determined minimum distance in the two sides of L).

P$_{\mathbf{strip}}$ is computed by simply scanning **P** and selecting the points within distance δ from L. If **P$_{\mathbf{strip}}$** is sorted in increasing order of x2-coordinate, then (ii) implies that each point in **P$_{\mathbf{strip}}$** only need to be considered to the next 7 points in that list (at most 4 points on the opposite side are relevant, but other 3 on the same side may appear between them in the list).



The overall algorithm first sorts **P** according to both coordinates to obtain an x1-list ordered by x1-coordinate, and an x2-list ordered according to the x2-coordinate. Using these, both steps (divide and merge) take O(n) time: dividing uses the x1-list to find the halving line and both sub-problems inherit the x1- and x2- order; merging uses the x1-list to get **P$_{\mathbf{strip}}$**, and then the x2-list as just described to determine the candidate pairs. We omit the detailed pseudocode for CP-Merge, which should be clear by now.

3. The implementation

1. Implement a sequential brute-force version of the problem accepting as command line parameter a filename containing the coordinates of the points, and another filename for the result. Example:

```
> closestpair original.dat result.dat
```

2. Write another program to randomly generate the points in 2D (or eventually in 3D) space. The file must contain also information about the number of points, the domain limits and the number of dimensions. Coordinates must be doubles.

3. Next you should implement a working MPI version accepting any number of processes.

For this version, you should try to estimate execution times and calculate performance indexes. Obtain also actual benchmarks using physical cores of processors on a computer.

4. Consider finally the described improvements (divide-and-conquer). Show some experimental results demonstrating your improvements.

4. Report

A report explaining your algorithms, results, and behaviors should be submitted with your code.