Relevant concepts

- *Reading from file*. We have learned in Assignment 0 different ways of Reading a file, for Assignment 3, the most efficient way to engage this task, is using fread(), given that we can read and store in a buffer big blocks of data, that will then be available in memory to be formatted and used in the code.

  We have used this approach to guarantee a good performance in the recursive Reading of the file, in the case that it´s size exceeds 1Mb of memory.

- *Locality*. closely related to what we discussed before, we load the biggest allowed "block" of data from the file and then we format each point into float to perform the mathematical calculations once we have it on memory, that gives a much faster results compared to formatting each point or line when it is read.

Intended program layout

The program splits into three subtasks: The reading of the file respecting the memory constraints, the formatting of the data, and the computation of distances between points storing the frequency of each of them.

This tasks will benefit from the use of parallelism obtained through OpenMP library.

Because of the nature of the Assignment, the way the reading is performed affects the way the calculations are done. Here we present the layout for reading and computing distances:

- Finding the maximum and optimal number of points allowed to load each time in order to not exceed 1Mb:
  we used 10000 for this parameter.
- Check if the file given has same or less number of points than the maximum allowed, because if that is the case the whole file can be loaded and the points compared with a simple double loop, taking into account that the distance between x0 and x1 is the same as the distance between x1 and x0 and should be counted once.
- If the file has more points than the allowed number, a modification has to be performed, the main idea is to divide the file in allowed blocks (blocks with a number of points equal or less to the maximum allowed), and have a number of fixed points that will be compared to the points in the first block, then with the points in the next block and so on until the end of the file, then the fixed points will advance until all the points have been compared. Note that as the loop advances the number of points to compare reduces.

As for the storing of frequencies the efficient approach is to create a vector which indexes are all the possible distances in the 3 dimensional cube of size 20x20x20 (from -10 to 10 in x ,y ,z). It is clear that the distances found will have to be converted to integers or more precisely multiplied by a factor of 10 in order to compare them with the indexes of the vector, the calculations are as follows:

- Max distance inside the cube = sqrt((20)^2 + (20)^2 + (20)^2) = 34.64 (2 decimal numbers).
- Min distance 0.01.
- Possible distances: 0.00 0.01 0.02 0.03 … 34.64.

According to this we create a vector V of size 3465, we format the points to make them integers, and each time we compute the distance between points we use it as index for V and augment the value stored in that index by 1.

Finally but probably the trickiest part of the code, is the efficient formatting of the data loaded from the file using fread(), as Suggested in the course web we took advantage of fixed format of the points in the file and the relationships of ASCII codes with their integer values as shown below: - We know that every coordinate will have a format like the next example: +01.330. - The points will be stored as char data type, and we know that the order of the values of the digits ASCII codes, is the same that the order of the digits themselves:

```
ASCII    digit
48       0
49       1
50       2
.
57       9
```

So if we subtract the ASCII value of ‘0(48)’ to the ASCII value of the other digits, we get the actual digit. And that is practically the smart trick to format the points in an efficient way. Being careful with the position of each digit in the coordinates, the points can be immediately obtained as integers: - the number after the sing will be multiplied by 10000 and will be added to the next number multiplied by 1000 … if we have +01.330 then = 0*10000* + *1*1000 + 3*100* + *3*10 +0 = 1330.

Note: for the parallelisation we only used para for with and without reductions, so it was a simple and straight forward way to attain good performance.