Assessment: Nearest and Furthest

Overview

This programming exercise requires that you complete the following tasks:

- Write a C++ program that, for each of a collection of objects in a 2d space, finds the distances to the nearest and furthest other object.
- Analyse the distributions of the nearest and furthest distances.
- Parallelise the code using OpenMP.
- Experiment with the parallelisation and comment on the speedup achieved.

You will need to submit your code and a report, including program timings and graphs of distributions.

C++ programming task

The basic code should be able to initialise a set of at least 100,000 object locations within a 1.0×1.0 square in each of the following ways:

- Randomly generate object locations, with both x and y coordinates taking values between 0.0 and 1.0.
- Read in object locations from a CSV file, where each line consists of an x coordinate, a comma and a y coordinate. All coordinates will take values between 0.0 and 1.0.

Simple wrappers for the C++ random number generation libraries and code for reading CSV files are both available on Blackboard. Feel free to use these if you wish, or to write your own code.

After initialising the locations of the objects, your code should determine, for each object, the distance to the nearest and the furthest other object. These distances should be written to two separate files, with one distance per line of output. Your code should also determine the average distance to the nearest object and the average distance to the furthest object, outputting these results to the console.

While there are sophisticated algorithms for performing these tasks, for this exercise, use the naive algorithm. That is, for each object, simply calculate the distance to each of the other objects in term and find the minimum and maximum values.

Two different distance measures

All of the objects are located in a square, with sides of length 1.0. You code should be able to use two different measures of the distance between two objects:

- The standard distance, calculated in the usual way.
- The shortest distance in a 'wraparound' geometry, where an object moving beyond an edge of the square results in it appearing at the opposite edge.

Figures 1–4 illustrate situations where the two geometries result in different values for the distance.

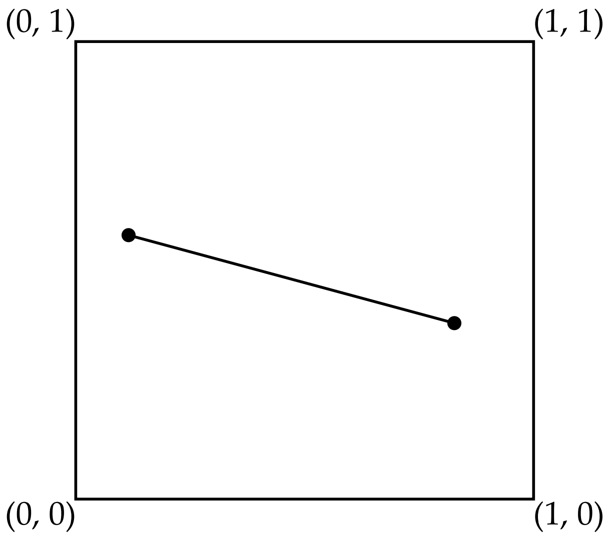

Figure 1: The shortest path between two objects in the standard geometry. Here, this gives a distance of 0.737.

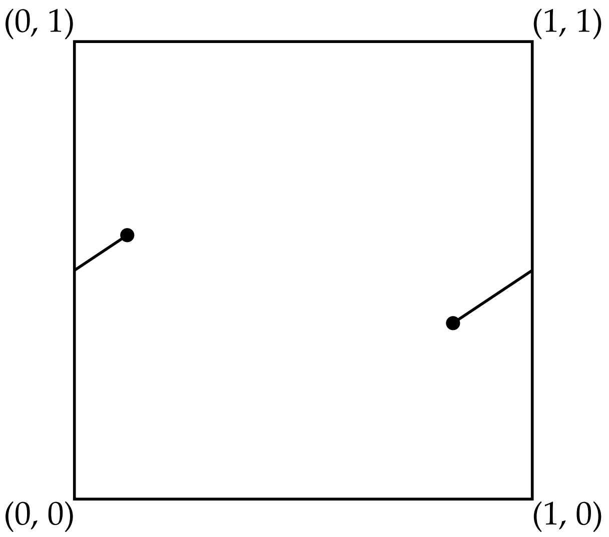

Figure 2: In the wraparound geometry, there is a shorter path, crossing the left edge and reappearing on the right. Here, this gives a distance of 0.347.

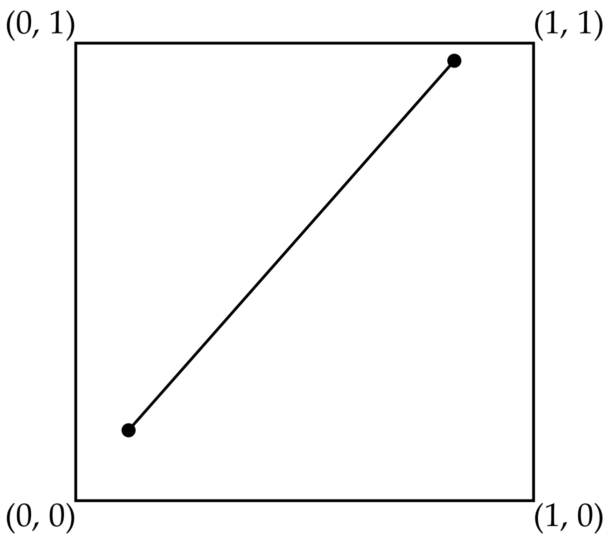

Figure 3: The shortest path between two objects in the standard geometry. Here, this gives a distance of 1.076.

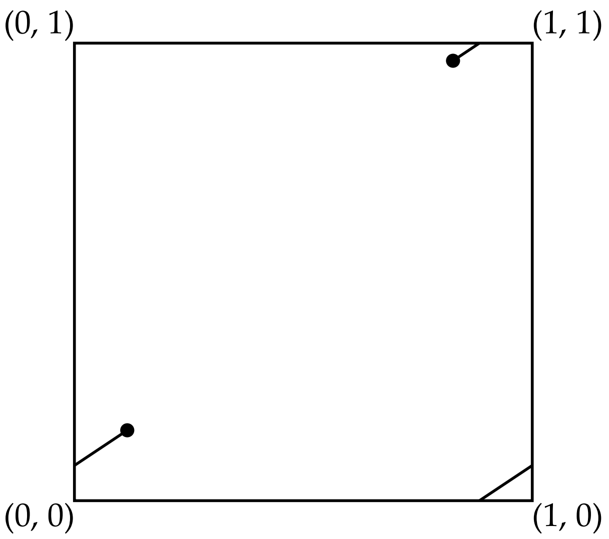

Figure 4: Again, in the wraparound geometry, there is a shorter path. In this case, it crosses from left to right and from the bottom edge to the top. This gives a distance of 0.347.

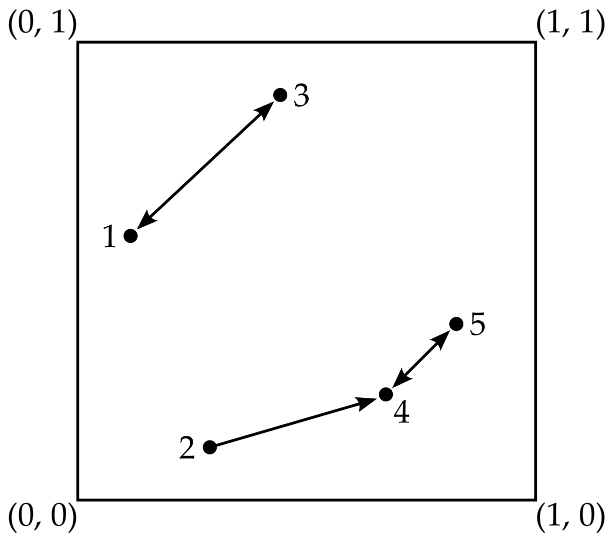

Figure 5: Arrows point from each object to its nearest neighbour, in the standard geometry. (Distances to nearest neighbour: 0.449, 0.402, 0.449, 0.218, 0.218. Average: 0.347.)

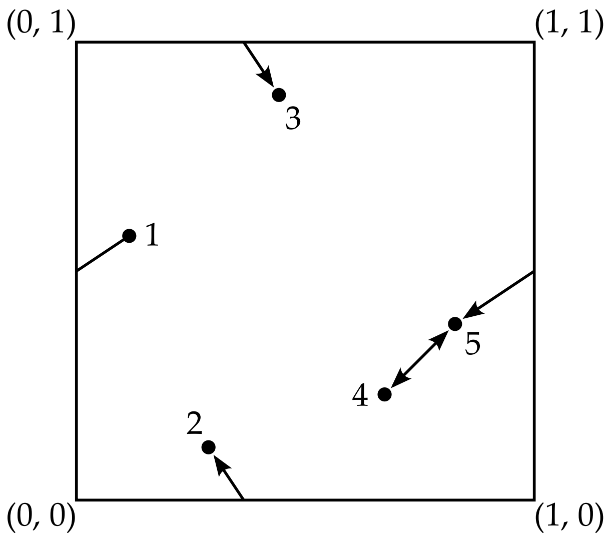

Figure 6: Arrows point from each object to its nearest neighbour, in the wraparound geometry. (Distances to nearest neighbour: 0.347, 0.277, 0.277, 0.218, 0.218. Average: 0.267.)

OpenMP programming task

Having written serial code to find the nearest and furthest distances for both of the two geometries, use OpenMP to parallelise it. Both the calculation of the lists of nearest/furthest distance *and* the calculation of the average nearest distance should be parallelised. Ensure that the parallel code runs correctly.

Comment on the effect of the parallelisation on the run time. How does this depend on the number of threads used. Would you expect the choice of loop scheduling to affect the time taken? Perform some experimentation with loop scheduling and report on the results.

Additional programming tasks (challenging)

You may have noticed that the naive algorithm described above requires the distance between each pair of objects, A and B, to be calculated twice, once when finding the distance of the objects closest to and furthest from A and again when finding the distance of the object closest to and furthest from B. Rewrite the serial code so that each distance is only calculated once. Then parallelise this code, ensuring that it still runs correctly.

Comment again on the run time of the various versions of your code. Experiment with the loop scheduling for the new version of the algorithm.

Report

Your submission for this exercise should include a report, briefly describing the operation of the code and presenting results in the form of graphs of the distributions of the nearest and furthest distances, for each of the two geometries. One way to produce these graphs would be to use Python to read in the data and Matplotlib to produce the charts. The report should also present code timings, illustrating speedups achieved through parallelisation. Your report should be no more than 4-5 pages in length.

What to submit

In addition to the report, submit your C++ code and any Python code you have used for plotting graphs. While the main C++ code may contain multiple files, it should be a single program. If you have tested multiple approaches, these should be implemented in different functions that can be called from the same main function.

While having correctly functioning code is of prime importance, you will also be assessed on the overall *quality* of your code, i.e. its structure, readability etc. Your ability to write a coherent report, with appropriately chosen graphs, will also be assessed.