Networked Software for Distributed Systems
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Project 3: MPI

Carlo Laurenti Argento
Gianluca Scanu
Federico Zanon

A.Y. 2023-2024

Contents

Introduction

Assignment

Scientists increasingly use computer simulations to study complex phenomena. In this project, you are to implement a program that simulates the behavior of a population of fish. The program considers N fish that move in a lake (modeled as a 3D cube) with linear motion and velocity v (each fish following a different direction). Each fish has a size. When two fish are close to each other and they are not of the same size, the biggest fish eats the smallest one and its size increases by one. The program outputs, at the end of each simulated day, the number of fish in the simulation, the size of the smallest fish, and the size of the largest fish. A performance analysis of the proposed solution is appreciated (but not mandatory). We are interested in studies that evaluate (1) how the execution time changes when increasing the number of fish and/or the size of the lake; (2) how the execution time decreases when adding more processing cores/hosts.

Assumptions and Guidelines

The program takes in input the following parameters:

- N = number of fish.
- L = side of the 3D cube where individuals move (in meters).
- v = moving speed for an individual.
- d = maximum eating distance (in meters): a fish that is closer than d to at least one bigger fish, gets eaten.
- t = time step (in seconds): the simulation recomputes the position of fish with a temporal granularity of t (simulated) seconds.

Technologies

The one and only technology used for the development of this project is MPI.

Realization

Design

When considering technologies for parallel computing and especially the MPI technology an efficient and distributed utilization of resources is paramount.
In this case, in order to maximize the employment, the computation is split among MPI processes. Therefore communication and synchronization are needed to produce coherent results at the end of the computation. These concepts act as basis for the idea behind this project and it is possible to identify them when looking at Figure 1.

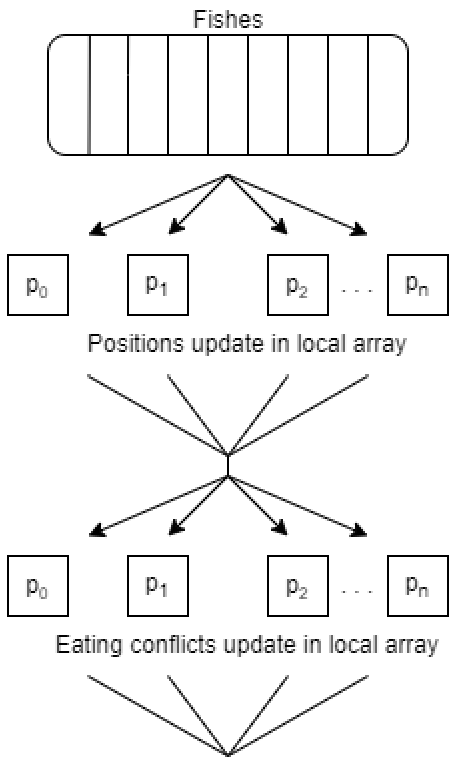
Fishes
p_0
p_1
p_2
. . .
p_n
Positions update in local array
p_0
p_1
p_2
. . .
p_n
Eating conflicts update in local array

Figure 1: Procedure idea for a single iteration

Here the population of fish is represented as an array, postponing for later the discussion of its actual implementation. Each process p_i is assigned a subset of fishes to handle throughout the procedure. This means that the computation for the position update at each time instant and the handling of the conflicts between close fishes of different sizes is carried out by separate processes. At the end of the procedure the whole array is once again reconstructed to let each process a view of the whole situation before the new iteration.

Implementation

The core of the implementation lies in the realization of the fish structure and its use within MPI primitives.

```
typedef struct{
    double x, y, z;
    double sx, sy, sz;
    int size;
    int eaten;
    int predator;
} Fish;
```

The fields of the Fish structure are:

- *x, y, z* represent the fish's coordinates in the lake;
- *sx, sy, sz* represent the fish's speed components;
- *size* is an integer representing the size of the fish;
- *eaten* is a flag indicating whether the fish is alive or dead;
- *predator* is an integer representing the fish which has eaten the considered one, if the fish is dead.

With the intention of facilitating the communication between processes this structure has been used to define a new MPI Datatype using MPI primitives. Therefore, each *Fish* has been encoded into an *mpi_fish_type* for the transmission of the local arrays. In this way each process has been enabled to manage fishes' arrays without abstracting their properties into integers, making updates and conflict resolution easier.
Finally, for what concerns communication and synchronization of the processes involved specific MPI primitives have been used to ensure a correct computation. In particular *MPI_AllGatherv* and *MPI_Barrier* have been employed respectively to store each local array received in the global one and to solve synchronization criteria.

Testing

For what concerns testing, the studies made aimed to show how the execution times changes when increasing specific parameters of the run. In detail the three plots represent respectively the software behavior with respect to changes in the number of fishes, in the size of the lake and in the number of cores employed. It is important to note that for each of these tests the same hardware was employed (an 8GB ram, i7 10th generation processor machine) and that for the case of core testing a limited number of executions were made due to high execution times. Moreover the results are what they were expected to be, showing that more fishes or a bigger lake increase the computational load and that work partitioning between different processors produces better behaviors in terms of time.

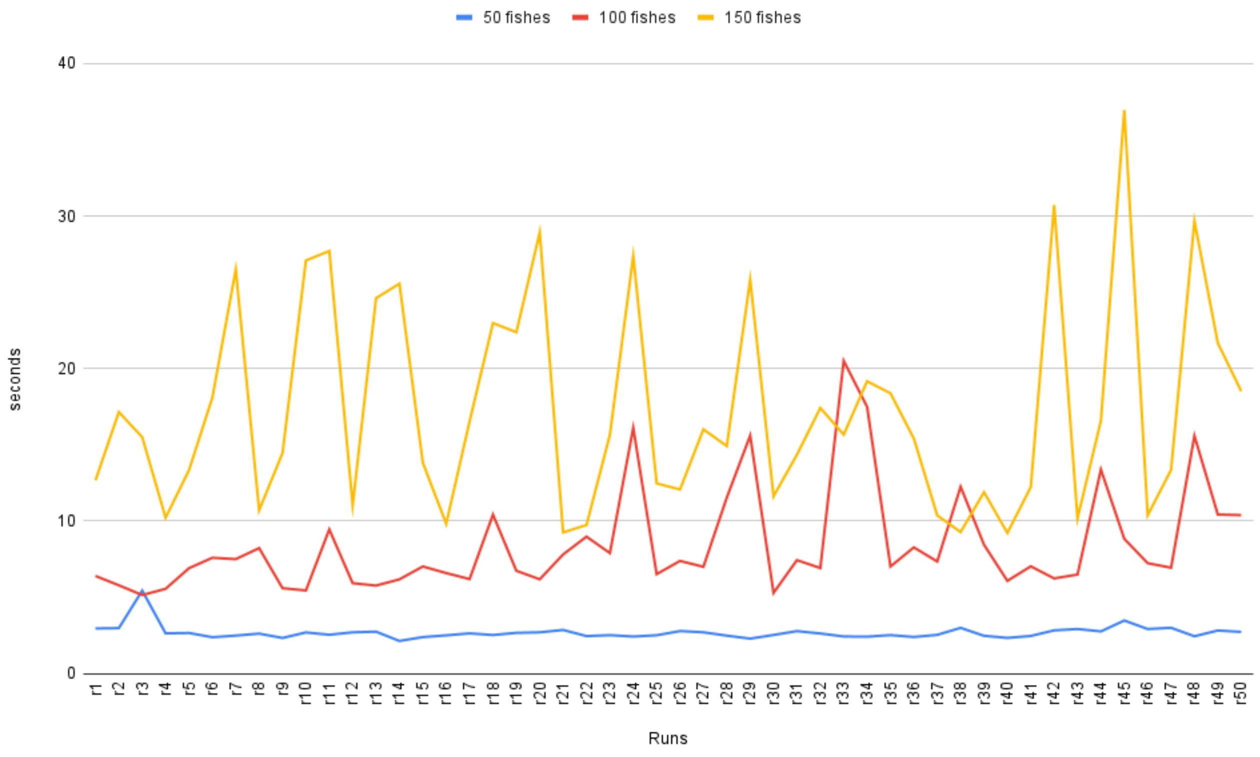

Figure 2: Execution time trend for increasing number of fishes

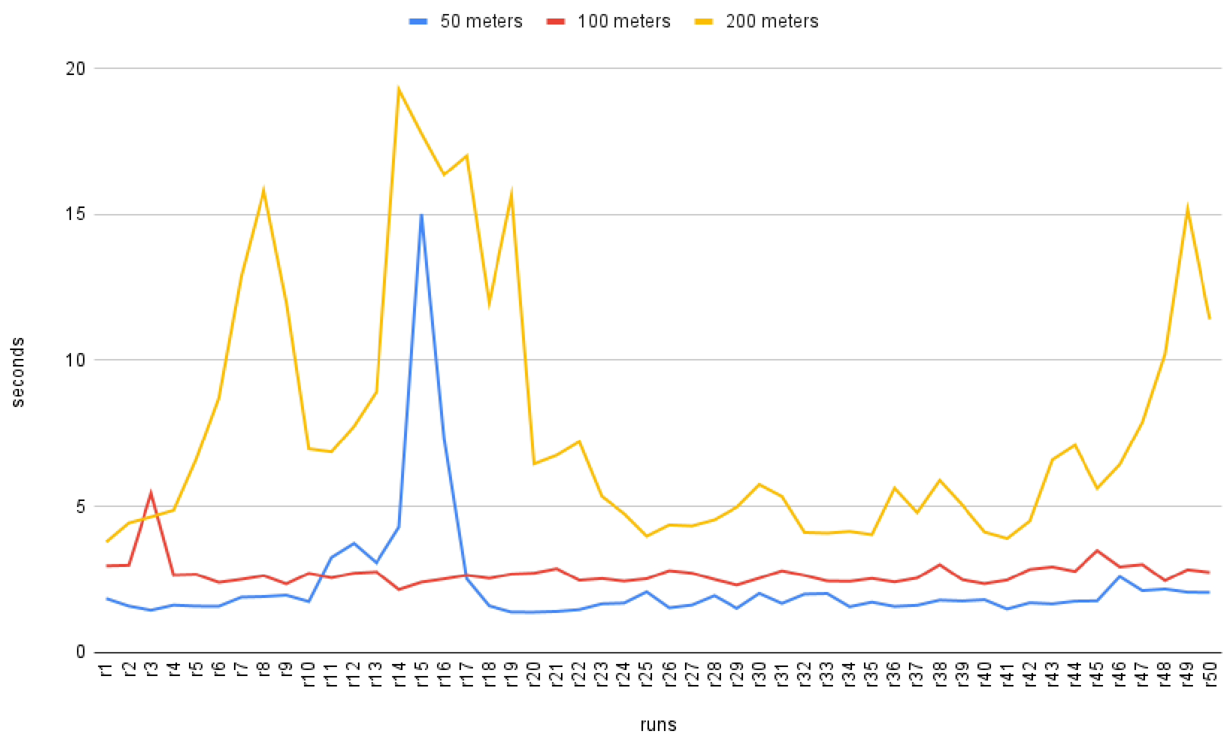

Figure 3: Execution time trend for increasing number of lake side sizes

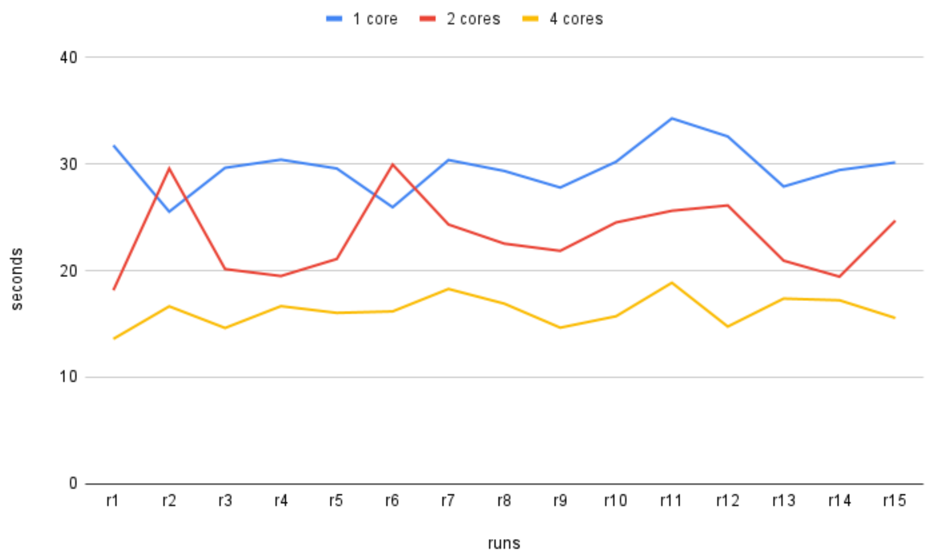

Figure 3: Execution time trend for increasing number of cores