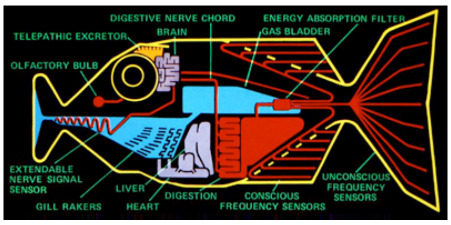
Parallel Programming Concepts WS 2010 / 2011



Assignment 3

(Submission deadline: Feb 19th 2012, 23:59 CET)

General Rules

Each of your code solutions has to fulfill the described application functionality (‚correctness‘) regardless of the underlying hardware. After finishing, your program must terminate with exit code 0.

Two out of four tasks must be solved correctly in order to pass the assignment. Your submitted solution must contain only a **single Scala source code file per task**, which compiles with Scala 2.9. Programs expecting any kind of keyboard input at any point in the execution are not allowed. Documentation should be done inside the source code. Students can submit solutions **in the established teams from assignment 1**.

Please submit your solution **before** the given deadline at https://www.dcl.hpi.uni-potsdam.de/parProg/. Incorrect submissions (missing source code, missing inline documentation) are rejected, with one possibility for re-submission.

Introduction

In this assignment, you should develop an implementation of the Wator simulation game in Scala 2.9.x:

http://de.wikipedia.org/wiki/Wator

http://www.scala-lang.org/downloads

The game rules should be implemented as described on the web page. The game field consists of a n x n grid of cells, modeling an ocean. Each cell can either contain water, a fish, or a shark. The initial distribution of sharks and fishes on the grid should be random. The simulation runs in rounds (‚generations‘). The evaluation order per generation is implementation-specific.

Fishes and sharks follow specific rules for their activity per simulation round. The grid should be understood as flattened torus, meaning that the upper border is connected to the lower border, and the left border is connected to the right border. It is therefore not possible to leave the ocean alive.

The goal is to maximize the number of generations being computed per second.

A fish first checks the surrounding cells in random (!) order, and moves to the first identified free neighbor cell. Every fish has an egg counter that increases by one with each simulation round. If a pre-defined number of eggs is reached, a new fish is born on the first identified free neighbor cell, and the egg counter is resetted. If no cell is free, no new fish is born, and the egg counter remains the same.

A shark first checks the surrounding cells in random order. If a fish is found on a neighbor cell, the sharks moves to this cell and eats the fish. If no food is available, the shark just moves to a free neighbor field. Every shark has a starvation counter, which increases with each round. If a pre-defined constant limit for the starvation counter is reached, the shark dies. New sharks appear under the same model as the fishes.

Both the fish and the shark egg limit and the starvation counter limit should be pre-defined constants in the code.

Task 3.1

Develop a command-line program in Scala, which implements the simulation in a serial fashion. The code should iterate over a ocean grid data structure and check each cell for its content and the appropriate activity.

Test your code with fixed initial distributions, e.g. were 1/3 of the ocean space is filled with fished, 1/3 with sharks, and 1/3 with water. Step through your simulation with very small sizes, to make sure that the rules are implemented correctly.

Measure the generation rate per second with different grid sizes / parameter constants and a random initial distribution.

Task 3.2

Modify your code so that the simulation parallelized with Scala:

http://www.scala-lang.org/docu/files/ScalaByExample.pdf

There are different parallelization strategies, for example:

• Each cell is modeled by an actor.

• Each fish resp. shark is modeled by an actor.

• Groups of cells / animals are modeled by an actor.

In order to coordinate the execution, implement a global barrier for all activities that marks the end of the current simulation generation computation. Measure the generation rate per second with configurations as in Task 3.1. What is the performance difference to the serial version with different parameter settings ? Think about the way how the simulation semantics change by the parallel implementation. Document your thoughts shortly in the code.

Task 3.3

Visualize the fish tank through Java graphics API, which is directly accessible from Scala code. An imperfect code skeleton with all the necessary Java Swing initialization is provided at:

http://www.dcl.hpi.uni-potsdam.de/teaching/parProg/wator_fragment.scala

Experiment with the synchronization between display rendering loop and simulation computation loop. Consider the problem that Java Swing component updates are only allowed from the original AWT event dispatching thread.

Suggestions and improvements for the provided code skeleton are welcome.

Task 3.4

Try to modify your parallelized solution so that no global barrier is needed, while the concept of simulation rounds is kept. This could, for example, being done by letting each cell keep a history of states from earlier simulation time stamps.