Project, Concurrency and Parallelism

Jean-Viven Millo
INRIA Sophia-Antipolis
jean-vivien.millo@inria.fr

Andrea Tettamanzi
UNSA
andrea.tettamanzi@unice.fr

November 23, 2012

Abstract

The goal of the project is to apply the notions you have learned during this course. In order to realize the project, you will need the following documents: i) this project subject ii) the course iii) the JAVADOC.

This assignment can be done either alone or in a team of 2 persons maximum.

1 Cellular Automata

The goal of the project is to implement Cellular Automata (CA) in a multi-threading fashion along with two algorithms that run on CA.

A cellular automaton consists of a regular grid of cells. The grid can be in any finite number of dimensions but, in the project, we will consider a CA of dimension 2. Each cell is a finite state machine. The state machine is the same for every cell but the initial state can be different from a cell to another. For each cell, a set of cells called its neighborhood (usually including the cell itself) is defined relative to the specified cell.

The state of the CA is the tuple composed of the state of each cell. An initial state (time $t = 0$) is selected by assigning a state for each cell. A new generation is created (advancing t by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously. *inspired from Wikipedia*

1.1 Software Engineering

Define the class diagram such that a CA is a grid of cells. The same class CA can be used whatsoever is the automata in the cells. The execution of the CA is multithreaded and each thread computes one to many cells. I recommend the usage of a class CA, a class cell and an interface automaton that will be implemented in order to realize an application.

1.2 Communication Between Cells

Most of the application based on CA requires that a cell is able to send its current state to its neighborhood. However, a cell does not have any reference to the other cells. A cell is identified by its coordinates in the grid and can send its state to another cell by asking the *communication manager* (which is part of the CA) to do it by specifying the coordinates of the addressees. There can be many of them! Of course, every cell can ask the communication manager to get the messages for which it is the addressee.

The communication manager has to run in a separate thread than the CA. Many cells can interact with it simultaneously but the access to the same resource should be synchronized.

1.3 Multithreading

Starting from the number of cores available on the computer and an estimation of the complexity of the update function of the automaton (simple, average, complex), one should find a rule to determine the appropriate number of threads to use in order to maximize the performances.

1.4 GUI

The CA should have a graphical representation. Each cell should appear as a box in a graphical grid corresponding to the topology of the CA. The boxes should contain information about the current state of the automata in the cells. One could define a *color code* associated to the possible states of the automaton. The GUI should also contain action button to start/pause/stop the simulation but also a *step* button that run a single step of CA. The delay between two steps could be parametrized. The initial state of each cell can be specified.

1.5 Persistence

At any moment during the execution of the CA, one should be able to save the current state of the CA in a file and then restore it later on. The format could either be the basic binary serialization format of JAVA or a defined XML format. In the first case, every class should implement the *Serializable* interface. In the second, the XML format should be defined by you.

1.6 Game of Life

Implement the interface automaton in order to realize Conway's Game of Life. All the information required to define the automaton and the update rules is available at this address.

1.7 Firing Squad Synchronization Problem

from Wikipedia The firing squad synchronization problem is a problem in Computer Science and cellular automata in which the goal is to design a cellular automaton that, starting with a single active cell, eventually reaches a state in which all cells are simultaneously active. It was first proposed by John Myhill in 1957 and published (with a solution) in 1962 by Edward Moore.

Consider a CA of one dimension with a finite size. The automaton of each cell is composed of a inactive state, some active states and a firing state. Every cell is inactive at the instant 0 and they remain inactive until a communication occurs. When the user decides, it trigs the left-most cell that goes in active state and start to send messages to other cells. Eventually every cell moves simultaneously and for the first time into the firing state.

The goal is to design the automaton (defining the number of active state, defining the transition between states, and the actions associated to these transitions) in order to implement the firing squad synchronization. The faster the firing occurs, the better.

The problem can also be considered for a two-dimension CA

2 Deliverable

The first deliverable is a synthesis report which presents the realizations of the project and an introduction to the implementation. The second deliverable is the JAVA implementation of the project.

The mark will be based on the following criteria:

- The clarity and usefulness of the report.
- The usage of the models of computation to express the complexity of your algorithm.
- The functionality! The program should compute a correct result.
- The usage of the JAVA primitives to do multi-threading.
- The clarity and simplicity of your implementation. A javadoc or at least some comments are expected in the source file.
- The handling of exceptions and the usage of exceptions. For example, if the two matrices have incompatible sizes, your code **throws** an exception such as IncompatibleSizeException.
- Your honesty! If you take a piece of code somewhere, say it and give credit.
- The ease of installing the implementation on another computer

An archive containing the report plus the source code has to be sent to `jean-vivien.millo@inria.fr` before December, 13th 2012 at 23:59 (French time).

The defense will take place on Monday, December 17, 2012, from 2 to 5 pm. Each team will be allowed 15 minutes (followed by 5 minutes of questions) to present their work. Each team is expected to prepare some slides and a demo.