Projects M1 2019-2020

Each of the following projects must be realized in groups, ideally by 4 students per project. You will work at USTH, during the six weeks dedicated to this TU. The first week is dedicated to documentation, and practise (Git, Qt, ...).

You will use Git to work together: first, each student will use its own branch pulling the dev one (pulled from the master branch). During a day session, you will commit regularly, at least once per hour, but more is better. At the end of each day, you will merge your work to the dev branch. At the end of each week, you will merge to the master branch.
The Git server have to be private, with an access granted to lilian.aveneau@usth.edu.vn.

You will have to synchronize inside each group in order to work efficiently! Using an agile method, you will met together at the beginning of each week to decide what to do, and at the end of each week to check the results and to understand what doesn't work as planned. Use Trello or Kanban to organize the task to do. Then, during the week you will grab some tasks:-)

Each week, we will have a visio to check the progress and the problems. You will defend your work at the end, the saturday of the last week. I will come at Hanoi to discuss and help the last week (first of March).

Notation will be made per group, and individually by checking the Git production.

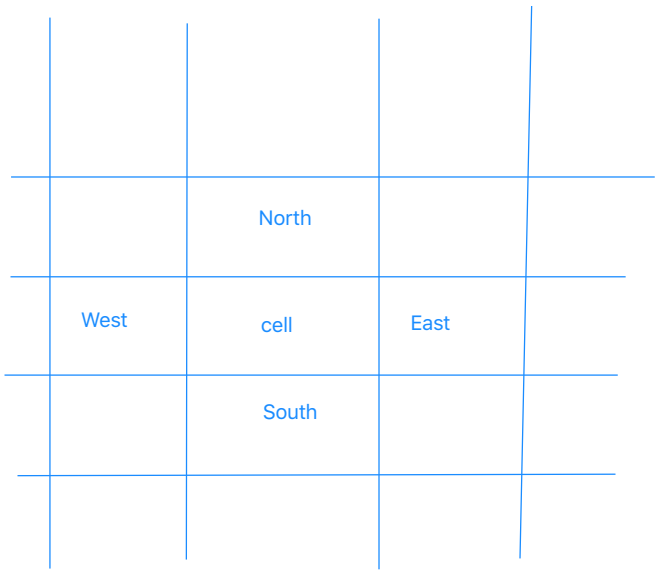
This year, four projects are open for subscription. The final assignments will be send to you soon. Give me your preference, by priority (first: the most preferred; second....).

1. **Cellular Automaton Simulator**

A cellular automaton (CA) is a regular grid of cells, each one in one of a finite number of states, such as **on** and **off**. For each cell, a set of cells called its

neighborhood is defined relative to the specified cell. A new **generation** is created

(advancing time t by 1) , according to some fixed **rule** (a given 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 neighboring.



Generally the rule is the same for each cell. When the rule is a stochastic function, nevertheless, its application may vary randomly, leading to **Stochastic Cellular Automata.**

The goal of this project is to write a software to **simulate this kind of CA**, called SCA.

More specifically, the constraints are:

- programming language is C++
- grapical interface made using Qt
- description of a SCA in XML
- the software must handle 2D space.

2. PRAM simulator

A **parallel random-access machine** (PRAM) is a shared- memory abstract machine. As it, it is a very interesting model to think about or to discuss parallel algorithm.

This model assumes two things:

- the **shared memory is infinite**, and its access from any parallel processor costs nothing;
- the **number of parallel processors is infinite**, and they are automatically synchronized.

These two assumption are unrealists, but allow to discuss parallel algorithms simply. The **synchronization at processor's level** is very strong, and directly linked to vector-machine (like MMX, SSE, AVX, ...).

The simulation of this kind of machine can be made at an instruction level, displaying the action in the memory. For instance, if we play with a linked list of 4 cells, then we can display the 4 initials values, then the values after the first parallel instruction, and so on.

The goal of this project is to write an application that **shows the evolution of the memory** (cells of a linked list, or of a vector) for a given algorithm. The constraints are:

- use C++ and Qt for the graphical interface;
- use XML for dealing with the algorithm to simulate, and its parameterization.

3. Reduce

Reduce is a parallel **design pattern** that consists to compute a value from a set of values. For instance, we would like to compute the sum of the vector [1,2, 3,4, 5], so 15. The result may be produced sequentially, or in parallel. The operation to apply must be commutative.

One big problem here is linked to the **floating point representation** of **real numbers.** Indeed, it is well-known
that the sum of two floats is rarely correct!

To obtain a "good" approximation of the result, many algorithms may be proposed. The goal of this project is to **compare some of these algorithms**, in **quality** and **complexity.**

The constraints are:

- use of C++ and Qt;

- use of XML for the description of the experiments, and to save them.

4. **Rendering**

In **computer graphics**, **rendering** means computing an **image** from a representation of a scene, including the geometry, materials, participating media, camera,... It is used widely nowadays, for film (Marvell, Aquaman, Deadpool,...), anime, and others applications.

There exist two family of methods:

- **offline rendering**, where the image is calculated on HPC for instance, in batch mode;
- **interactive rendering,** for instance for games or website.

The goal of this project is to write a **modular software for interactive rendering,** allowing to implement and test new algorithms.

The constraints are:

- use of C++ and Qt;
- modular architecture allowing to add new algorithms (like gimp for instance);
- OpenGL rendering with Compute Shaders,...
- XML file format for scene description.