Simulation of complex systems

The goal of this project is to develop a parallel tools to simulate the behavior of a group of particles. You will have to write a code that perform the simulation and then the corresponding parallel code.

Your code should be optimized and commented. You may work in team of two.

1 Flocking birds and schooling fishes

In this section, we consider the motion of a variety of particles that follows a simple set of rules. We will refer to this motion as the flocking behavior.

1.1 Introduction

Flocking is a generic term that describe the motion of a group of agents that follows a motion strategy designed to hold the group together. Such behavior is exemplified by the motion of groups animals, such as birds, fish, insects, and other types of herding animals.

In flocking it is assumed that the all the agents are following the exact same rules and apply the exactly the same algorithm. The only difference between the agents are their relative positions. Even with simple rules complex formations can be observed.

The spontaneous generation of complex behavior from the simple actions of a large collection of dynamic entities is called emergent behavior. As we will see in the next section, the algorithms that describes flocking behavior do not involve very sophisticated models of intelligence. However, even simple variants of this method can be applied to simple forms of crowd motion in games or motion pictures.

1.2 A simple model: boids

C. W. Reynolds wrote one of the most used model for flocking behavior in 1986 with his work on "boids". In this system, the behavior of each agent is regulated by the combination of four simple rules. In order to give a mathematical description to the motion of boids, each boid is assigned a position $x \in \mathbb{R}^3$, a velocity $u \in \mathbb{R}^3$ and a mass M.

1. Separation: Each boid tries to avoid collisions with other nearby boids. Each boid emits a repulsive potential field whose radius of influence extends to its immediate neighborhood. Whenever another boid gets too close, the force from this field will tend to push them apart. Each boids try to move away from the local center of mass of the group. The rule writes

$$u_i = u_i - \frac{w_s}{Mn_s} \sum_{j=0, j\neq i}^{N} (||x_j - x_i||_2 < r_s) \cdot (x_j - x_i) \tag{1}$$

   where w_s is the weight of the separation force and r_s its the radius.

2. Alignment: Each boid's direction is aligned with nearby boids. Thus, local clusters of boids will tend to point in the same direction. The rule writes

$$u_i = u_i + \frac{w_a}{Mn_a} \sum_{j=0, j\neq i}^{N} (||x_j - x_i||_2 < r_a) \cdot u_j \tag{2}$$

   where w_a is the weight of the separation force and r_a its the radius.

3. Cohesion: Each boids tend to be drawn towards the center of mass of the flock. Of course, the boid cannot know where the center of mass is exactly since it is implicit and hence, the center of mass cannot be computed exactly. The rule write

$$u_i = u_i + \frac{w_c}{Mn_c} \sum_{j=0, j\neq i}^{N} (||x_j - x_i||_2 < r_c) x_j \tag{3}$$

   where w_c is the weight of the cohesion force and r_c its the radius.

4. Avoidance: Each boid will try to avoid the collision with obstacles. In order to simulate avoidance, the obstacle will generate a repulsive potential field. With such a field, as a bold approaches the object, this repulsive field will deflect the boid from its flight path, thus avoiding a collision. Avoidance can also be applied to predators.

1.3 Implementation

In order to advance the position of the boids, we need to solve an initial value problem for each particle

$$\frac{dx}{dt} = f(x,t) \tag{4}$$

In the forces mentioned in the previous section, the integration was done using Euler method, but higher order methods can be used in order to improve the stability of the simulation, especially for a large number of particles.

For the efficiency reasons, we may not want to compute all the interactions between all the particles since it will take $O(n^2)$ time for each time iteration. In order to avoid this cost, the particles may be stored in a grid or a quadtree (octree in 3D).

2 Project guidelines

The project you will be working on aims at creating a parallel version of a simulation code. A sequential implementation is provided as a reference for your work. You may write your own sequential code in *C*, *C++* or *Fortran*.

2.1 Analysis

Your first task is to perform a comprehensive analysis of the problem. The aim of this analysis is to understand and identify several aspects of the problem:

1. what are the main data of the problem.
2. what are the spatial dependencies between the data.
3. what are the time dependencies between the data.
4. how does the dependencies change during the simulation.
5. how can the data be clustered.
6. how are the data stored.
7. what is the complexity of the algorithm with respect to the number of data, to the configuration.

The analysis may have some quantitative aspects. In order to fully understand what how the simulation work, you should run the program with different parameter values.

2.2 Code design

Once the analysis is completed, the next step is to identify where the parallelism can be use efficiently. In this step you should decide how the data will be grouped and how you will store the data.

This step should give you a better picture of what computational step will be performed by each thread.

The next step is to map each thread to a process depending on your approach.

Up to this point, no programming is required.

2.3 Writing the parallel program

In this phase, your goal is to write a parallel version of the code using the parallel paradigm of your choice. The first version does not need to be optimal. You should write your code incrementally. For example, if you choose *OpenMP* as a parallel paradigm, you may start by using the directive ***#pragma*** *omp parallel* ***for*** around the ***for*** loops of the sequential implementation. But this should **not be** the final version.

During the design phase, you may have overlooked some parallel constraints. You should try to identify those as early as possible. For instance, if you choose *MPI* as a parallel paradigm, writing to the solution file should be done sequentially by all the process even if the file system is shared.

> **Remarque**
>
> If you choose to work from the provided sequential implementation, you may modify any class or function.
>
> There are several way to obtain a good parallel code. It will mainly depend on the data structure you choose to store the data.

2.4 Testing your implementation

During the development process, you should test all the aspects of your program. The main focus should be to improve the performances.

In order to evaluate the speedup of your implementation, you must choose a problem large enough that it takes a lot of time to be solved on one processor and increase the number of processor progressively.

2.5 Writing a report

The report is required. It should contain

1. an introduction explaining the purpose, objectives and content of your work.
2. a clear description of your work and of your design decisions.
3. a clear description of your results.
4. a conclusion.

Your report may contains graphs, tables and figures.

3 Evaluation

Your work will be evaluated on the following aspects

1. Quality of your program.
2. Performances of your application.
3. Analysis and relevance of your results.
4. Initiative.
5. Quality of your report.

You report should be in pdf format and your code commented.

References

[1] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[2] Mark Joselli, Erick Baptista Passos, Jose Ricardo Silva Junior, Marcelo Zamith, Esteban Clua, Eduardo Soluri, and Nullpointer Tecnologias. A flocking boids simulation and optimization structure for mobile multicore architectures. *Proceedings of SBGames*, pages 83–92, 2012.

[3] Chiang. Emergent Crowd Behavior. *Computer-aided Design and Applications*, 6, 2009.