# Parallel and Distributed Computing Report

Diogo Paulo Dias 93980
Gonçalo Pires Teixeira 94064
Tiago Bartolomeu Luiz 93976

March 2019

## 1 Implementation

Having implemented the serial version of the project, it was analyzed the dependencies between the operations. It was identified that there are a sequence of operations that must remain the same, more specifically, it must commence with the initialization of the particles followed by the creation of the array containing the cells, and just then the simulation can start. As specified in the project's statement, the sequence of operations can not be changed, it has to always be firstly computed the center of mass of each cell, and just then it can be computed the gravitational force applied to each particle and its new velocity and position.

### 1.1 Initialization of the Structures

On the beginning of the program, it was allocated the necessary memory to maintain the structures that support it, this includes an array of particles and an array of cells, where each one represents a center of mass. Instead of mapping the grid to a matrix, with a double indexing access (i.e., double array notation [ ][ ]), it is mapped into a single dimension array whose indexing is given by the following formula:

$$particle.cellX * ncside + particle.cellY$$

All allocated structures were obviously freed to prevent memory leaks in the program.

### 1.2 Computation of Center of Mass

The operation responsible for the computation of the center of mass of each cell iterates over each existent particle, without having the need for a specific order. Having said that, derived from a design choice where the cells do not have an array with their particles, having instead each particle the coordinates of its respective cell, we could not ensure that two threads are not accessing and

updating the same cell at a given instant. Memory synchronization is then a problem that needs to be taken care of. Two solutions were found, implemented and tested, so that their efficiency (i.e., runtime) could be measured, the first one being the usage of atomic operations on each update over a cell, and the second one the usage of a for reduction strategy. Using the sixth example of the "Project" page in Fenix (*arguments: 8 5 1000000 50*) to obtain comparable results, with 4 threads and 3 consecutive runs, the first solution has an average runtime of 15.019 seconds whereas the second averages 12.727 seconds. Given the 2.292 seconds difference between both times, the second solution was implemented.

A similar approach is implemented on the computation of the global center of mass, since it has the same problem, hence, the same solution.

### 1.3 Gravitational Force, Velocity and Position

The particle's new position is dependent of the force being applied on itself, so these operations need to be made in a sequential pattern. But the computation of a particle's new position is independent of the calculation of a new position of another particle. This is facilitated because of two characteristics of our implementation. The first characteristic is the coordinates of the cells where each particle resides. These are stored in each particle's structure, making the update to the cell coordinates "local" to itself. The other characteristic is when we change the coordinates of one center of mass. Because the sides wrap around we create a local center of mass, thus not changing the global state of the cell's center of mass. With these features, we divide the array of particles by the number of threads and each part goes to one thread, not being necessary to add sync instructions.

## 2 Results

Since we are not on an ideal system with an immaculate implementation, it is expected that the speedup is near the number of threads that are being used, so if we have 4 threads we expect something around 3,5.

The serial version, having as seed 8, a grid size 5, with 1000000 particles and 50 time-steps runs in 45.481 seconds. On Table 1 it is available a comparison between the time obtained in the parallel version with different number of threads, and their respective speedup when compared to the serial version. The computer used to obtain these results has 4 physical and 8 logic CPUs with 2.8 GHz, 16GB RAM and runs the *macOS* operative system.

Other examples were also tested, more specifically, the ones on the project's statement. The measured speedups also match the ones presented above (i.e., near the number of threads used, except with 8, which has a speedup between 4.5-5).

| **Number of Threads** | **Runtime** | **Speedup** |
|---|---|---|
| 1 | 46.568 | 0.9767 |
| 2 | 23.877 | 1.905 |
| 4 | 12.727 | 3.574 |
| 8 | 9.82 | 4.631 |

Table 1: Results