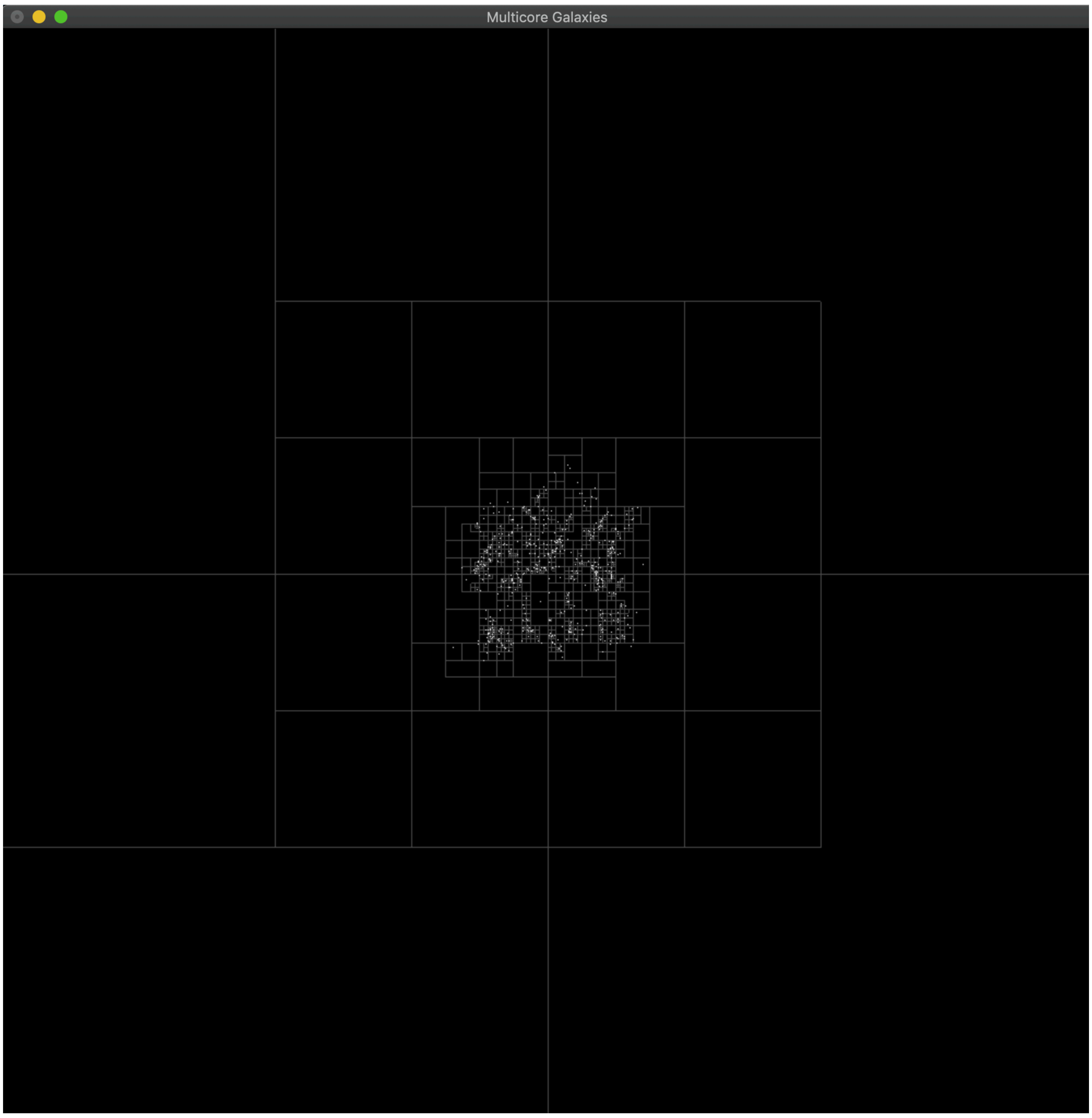
COMP426 MulticoreProgramming Project 2 report



Clément Péau
40102553

Foreword

I must point out that, I have been unable to turn in the first project due to fact that I contracted food poisoning and had to rest for 2 week straight. Thus I won't be able to compare the Threadpool approach to the TBB one.

Project Overview

In this section, I will detail the project and its global architecture.

We were tasked of creating a N-Bodies simulation using the Barnes Hut Galaxy Simulator algorithm. In a few words this algorithm enables us to make groups of stars to compute the attraction to other stars instead of looping over every stars and computing its attraction to every other stars. This computation being ressource intensive and because its the point of the assignment, I implemented Intel's multiprogramming library : Intel Thread Building Blocks(TBB).

TBB is a task focused parallelism Framework, meaning that we don't have a control thread, we only give task to the scheduler. This scheduler will make sure that every available thread will be used in the most efficient way and will spend the less possible time in an idle state. For this project we chose to have a maximum of 300 concurrent threads running. The program is set to run as a constant 30FPS.

My implementation

I have a class RootQuadrant that represents a galaxy, and contains every quadrant for our Barnes Hut algorithm. In those quadrant are recursively stored the stars. I chose to instantiate my stars as shared_ptr, this way in addition to the quadrant, I kept a strong link of each star inside a vector. It enables me to keep looping over the stars without having to go down the quad tree.

I chose to limit the depth of my quadrants to one pixel or more. Because If two stars were to be on the same x and y position they would have to divide the quadrants so much that with the branching factor the algorithm would be way slower than it is now.

In the main method I chose to instantiate a task_group and give it as a parameter to my Display class and my RootQuadrant. This way both will share the same thread pool.
The task_group also makes sure that the main thread will idle until all the threads are joined.

Task Parallelism

I chose to have my display and my computing unit running in parallel. Both are using the same RootQuadrant and Star list. Due to the high number of concurrent thread, I made sure that synchronisation would occur only once per cycle. A mutex is contained inside the RootQuadrant instance, the computation thread locks it, computes the values, apply them, then unlock the mutex. The Display thread then locks it, draw on the window and unlocks it.

Data Parallelism

The main gain of performance was due to Data Parallelism. The computation method used two for loops to calculate the acceleration of each star, then apply it. Using a system of cache, I was able to make sure the data was not subject to change, so I used a parallel_for for both the computation and applying the cache. The speed up was really significant since the computation method was the bottleneck of the program and was now way more manageable.