**Parallel Computing (CC4014)**

**Ecosystem Simulation**

Daniel Rangel – up201908562

16/12/19

1. **Summary**

For this assignment it was proposed for us to develop a sequential program that is to be implemented along with parallel programing techniques in order to make it run in parallel using different threads or processes. The program is meant to simulate an ecosystem of rabbits and foxes where they try to multiply and feed on each other.

In order to parallelize the program, I chose to use solely OpenMP, which to me had the easiest and quickest implementation as well as also being the focus from classes. As expected, I started by developing the program as a sequential one to later parallelize it. Throughout the development it was kept in mind to make it easier and “compatible” with OpenMP. Therefore, the main data structures used were 2 arrays, one of which two-dimensional that held the position of the different elements of the ecosystem that is used as a map for the different animals to know their surroundings, the second array was simple one-dimensional array that holds all the existing animals (dead and alive).

This was done so that later in development I could take use of the *#pragma omp parallel for* clause to parallelize the bulk of the program.

All the animals were structured under the struct “Object” that holds all the relevant information for the animals as well as some extra functions that should be common for all animals, such as *NewObject(), InitObject()* and *PrintObject() and PrintObjectList()*.

Some extra functions were also implemented but since they were specific for the animal in question (rabbit or fox). These functions will control the “Turn” of each animal where they move, procreate and hunt when it comes to the foxes.

Regarding the parallel aspect of the program, this was done by parallelizing the “turn” of all animals but considering the critical area which is when any animal needs to procreate which can only be done one at a time.

**Performance**

Unexpectedly after parallelizing the program I didn’t any improvement of the execution time of the program throughout the tests that were run. The main assumption I’m making that might have caused this is the data structure used to hold the ecosystem animals, the one-dimensional array. Still, these were the times taken from running the program, excluding input and output. The execution times for the program were taken from my machine with a 4.4Ghz 4 Core,8 Thread processor and 16GB of RAM.

Input5x5:

|  |  |  |
| --- | --- | --- |
| NUMBER OF PROCESSORS | EXECUTION TIME (seconds) | SPEEDUP |
| 1 | 0.000062 | 1 |
| 2 | 0.000088 | 0.70 |
| 4 | 0.000103 | 0.60 |
| 8 | 0.000161 | 0.39 |
| 16 | 0.001594 | 0.04 |

Input10x10:

|  |  |  |
| --- | --- | --- |
| NUMBER OF PROCESSORS | EXECUTION TIME (seconds) | SPEEDUP |
| 1 | 0.012209 | 1 |
| 2 | 0.014052 | 0.87 |
| 4 | 0.014603 | 0.84 |
| 8 | 0.027486 | 0.44 |
| 16 | 0.034910 | 0.35 |

Input20x20:

|  |  |  |
| --- | --- | --- |
| NUMBER OF PROCESSORS | EXECUTION TIME (seconds) | SPEEDUP |
| 1 | 6.849102 | 1 |
| 2 | 7.022108 | 0.98 |
| 4 | 7.123411 | 0.96 |
| 8 | 7.779259 | 0.88 |
| 16 | 7.212273 | 0.95 |

The execution times for the inputs 100x100 and greater took too long. I assume that my implementation of the second data structure becomes too big taking too much time to run through to execute the turn for each animal.

1. **Main difficulties**

There weren’t many notable difficulties while developing the program other than the main issue with the data structure I used for the animals that ended up costing the entire performance of the program. This is the main issue with the program that should be fixed, even though I was able to in time.

Other than that, a few difficulties arose from the implementation of OpenMP but these were quickly fixed by referencing the lecturers notes and materials.