APSD 19/20 - Final Project

Davide Caligiuri, 189682

June 2020

1 Introduction

This is the final project report for my parallel computing project, part of the final exam for the Parallel Algorithms and Distributed Systems course, lectured by prof. William Spataro at Università della Calabria (UniCal).
It consists in the implementation of a cellular automata (a variation of the WaTor cellular automata from literature) in C/C++ language, using parallel computing libraries, specifically MPI and OpenMP.

2 Problem description

Our cellular automata models the behaviour of three different populations: shrimps, fishes, sharks, each with his characteristics regarding lifespan and feeding habits. Though there are specific rules for each species, they can be summarized as:

- Each species has a defined lifespan and breeding phase;
- Fishes and sharks are carnivore, while shrimps aren't;
- **(personal addition)** Fishes and sharks can resort to cannibalism when reaching the breeding age, but they'll never eat their infants (to preserve population growth and avoid extinction due to self-consumption);

3 Technical information

All tests have been conducted on a Lenovo Thinkpad X230 with a Intel i5 QuadCore CPU supporting HyperThreading and 16GB of RAM: thus, the performances are to be taken as indicative.
All tests have been made using the latest versions of MPICH and OpenMP; care has been taken in trying to exploit affinity as much as possible through environmental variables and MPI directives, and has been proven to be a core element in obtaining reasonable results.

4 Performance

To note:

- Values in the table are execution times over 100 iterations on a NxN 2D grid, where N is the column index.
- The (total) number of threads must be less or equal to 8 (2*cores) due to HyperThreading; the maximum number of processes is instead bound to the number of pyhisical cores (4).

Pure OpenMP							
Threads	**1000**	**2500**	**5000**	**7500**	**10000**	**12500**	**15000**
1	3.002	18.73	74.84	168.4	300.2	469.2	675.9
2	2.457	15.27	61.08	137.4	244.3	382.7	551.4
3	2.455	15.28	61.04	137.5	244.2	381.8	550.8
4	2.456	15.28	61.06	137.4	244.0	381.7	550.4
5	2.457	15.28	61.45	137.5	244.0	382.0	550.7
6	2.471	15.28	61.30	137.6	244.4	381.9	550.9
7	2.456	15.27	61.04	137.7	244.2	381.5	550.5
8	2.454	15.28	61.07	137.7	244.0	382.0	550.8

Hybrid MPI/OpenMP (mpiexec -n 2)							
Threads/Process	**1000**	**2500**	**5000**	**7500**	**10000**	**12500**	**15000**
1	1.964	12.18	48.73	109.9	195.6	304.8	439.1
2	1.578	9.741	39.06	87.93	156.2	243.7	351.6
3	1.585	9.771	39.55	87.93	156.4	243.5	351.6
4	1.592	9.751	39.13	87.96	156.8	243.6	351.2

Pure MPI							
Processes	**1000**	**2500**	**5000**	**7500**	**10000**	**12500**	**15000**
1	3.002	18.73	74.84	168.4	300.2	496.2	675.9
2	1.964	12.18	48.73	109.9	244.3	304.8	439.1
3	2.238	13.96	55.57	116.8	244.2	346.4	466.9
4	1.778	10.95	43.73	100.6	244.0	277.6	401.5

5 Conclusions

As shown, the MPI version of this software scales reasonably, while the hybrid and OpenMP show a small improvement from the use of 2 threads, and way less so with more of them; this can be related to a variety of reasons, from the overhead due to synchronization between the two libraries, to the memory allocation and cache usage (which can become a bottleneck); on the last topic, correct usage of thread affinity features in both MPI and OpenMP has shown to improve performance significantly, and it is surely an important factor to consider in eventual future improvements. Also, it is possible that OpenMP impact can increase with the problem dimension.