Parallel and Distributed Computing

Academic Year 2021/2022

MPI Implementation

Group 23

Filipe Corral 81346 João Gonçalves 85211 Jiaxin Jiang 101925

Approach used for parallelization:

As for the OpenMP implementation, it is not possible to parallelize the generations, since each one depends on the results from the previous generation. Also the black sub-generation can only be computed after the red one finishes.

Therefore the data was parallelized when processed in each sub-generation. This way the map was equally divided between each of the nodes. In the end each nodes calculates the number of rocks and animals in the map and sends to the main node, that sums everything and prints the result.

To help the parallelization the *moved* flag, that indicated when an animal had moved from one iteration to the other was also changed. Now, this flag will indicate if a cell has been altered. This means that if an animal moves, It will be marked has having moved, but the cell he was previously in will also be marked as having been changed.

Decomposition used:

Again following what had been done in the OpenMP implementation, the map was divided by rows. Each process keeps the map division that was assigned to it, and one more line for each one of the borders (the last line from its neighbours). In addition two more shared lines for each border (the last two lines from each neighbour) are kept temporarily in each node to facilitate the conflict resolution process. These shared lines are the only ones that have to be sent by the nodes in each iteration, this way we minimize the memory being used and the number of communications between the nodes, making the map division more efficient.

The map could also be divided in blocks, and that strategy was discussed but, since the map sizes are very large, the improvements in speed were not considerable for the complexity of this solution.

Synchronization concerns:

In order to reduce the amount of synchronization needed, the shared lines described before were used. This way only those lines are synchronized between each sub-generation, and the conflicts resolved after.

Load balancing:

As for the OpenMP, since the arrays are divided equally between the nodes the load on each one should be approximately the same, but in a scenario where the animals are not evenly distributed in the map some nodes might have a higher workload.

MPI+OpenMP:

A version with MPI and OpenMP was also developed. For this version, since each node as an equal portion of the map, that portion in divided equally by the threads. The logic used was similar to the one described in the OpenMP implementation.

What are the performance results:

The results obtained in the lab computers for the serial and OpenMP (with 4 threads) are presented in table 1.

	Serial updated	**OMP (4 threads)**
r300	22.4	14.4
r4000	100.9	64.5
r20000	285.8	154.9
r100000	83.7	26.1

Table 1 – Performance results in seconds for Serial and OMP

In tables 2 and 3 we have the results for MPI and MPI+OpenMP, also computed in the lab.

	MPI (2 nodes)	MPI (4 nodes)	MPI (8 nodes)	MPI (16 nodes)	MPI (32 nodes)	MPI (64 nodes)
r300	20.9	19.5	13.1	5.3	5.8	1.1
r4000	93.3	63.8	39.8	22.6	14.7	12.6
r20000	337.3	148.8	84.3	64.8	39.9	35.2
r100000	79.3	47.0	82.5	114.8	99.7	136.8
r50	-	-	-	-	-	9.5

Table 2 – Performance results in seconds for MPI

	MPI OMP (2 nodes)	MPI OMP (4 nodes)	MPI OMP (8 nodes)	MPI OMP (16 nodes)	MPI OMP (32 nodes)	MPI OMP (64 nodes)
r300	21.6	30.0	16.5	4.0	2.1	1.3
r4000	121.2	71.2	38.1	21.9	14.5	12.8
r20000	255.6	151.4	91.5	71.1	39.8	35.7
r100000	102.7	106.0	68.6	81.2	77.9	108.1
r50	179.4	113.6	76.7	38.2	14.1	9.8

Table 3 – Performance results in seconds for MPI and OMP

As expected the results from the MPI are increasingly faster as the number of nodes increases, surpassing the ones from the OpenMP. With 4 nodes, the same as the four threads used, the results are slightly faster for the MPI. Since each node as all the information needed there are no memory conflicts, making the computation faster. The gap between the times is not larger because, contrary to what happens in OpenMP, we need to take into account the fixed communication time between each node that happens between each sub-generation.

It was also noticed that different from the rest, the times for *r100000* were slower as the number of nodes increased. The explanation for this can be that the map is relatively small (200x500) so the number of rows for each node is not sufficient to compensate the increasing amount of communication times. Also, this example has an enormous number of iterations (100000), so the increasing communication times are summed for all those iterations. The solution is therefore much better for huge maps with few iterations, like the *r50* (30000x8000 map).

The MPI+OpenMP times are sometimes worse than the MPI ones, being slightly better sometimes. This was not expected, but can be explained by the overload on the lab computers. This way the computers might have some threads already being used by other groups, and therefore run the OpenMP with very few, or even one thread, most of the time, making the times even worse than the MPI, since it was not worth doing the process of parallelization.

Due to a failure in the servers it was not possible to obtain the results for the *r50* example with the MPI implementation.