Parallel and Distributed Computing

Squirrels & Wolves

Second Project Delivery

Authors
66950 António Goulão
66952 Beatriz Mano
66995 Hugo Correia

December 6th, 2013

1. Approach used for parallelization

When parallelizing our serial version using MPI, our biggest concern was to guarantee a non-dependent computation between processes and, simultaneously, to minimize the number of communications between them.

2. Decomposition used

Since we distribute the world matrix among the processes a priori, the data decomposition is implicit, i.e, each process computes its own data.
The input division among the defined number of processes is calculated according to the following formula:

Chunck size = world size / number of processes lines ,

where the initial process (process zero) is responsible for *chunk size + (world size % number of processes)* lines.

The input is also distributed while is being read, making possible to the program to compute input files of huge sizes.

3. Synchronization concerns

The only synchronization concern regarding the MPI version is the communication, once processes must exchange information between them.

The more naive approach would be to send to each neighbouring process the information of the movements when a process would compute the frontier(s) line(s) of its chunk.

Our approach, on the other hand, is to append to each chunk a ghost line (one for each of its possible boundaries) which enables the process to exchange movements between two adjacent processes only once between sub-generations. This solution has, however, a tradeoff: the data size exchanged per communication is greater than it would be on the naive approach. Nevertheless, the overhead of establishing several communications would cost more than exchanging bigger messages per communication, as implemented.

4. Load Balancing

Load balancing wasn't our major concern during development since the only data structures we use are matrices and small arrays, which are even smaller now that each process has a smaller matrix to compute.

There could be some performance problems if entities are placed at a corner of the matrix leaving other regions of the unpopulated matrix, meaning that the load balance between processes won't be even, leaving some processes with a greater workload than others. This is a plausible scenario although we expect entities to be evenly distribute through the matrix.

5. Memory Requirements

Each process stores all the information about its *chunk* lines of the world matrix. The master process (process zero) takes a *chunk + world size % number of processes*. This process division makes the program more efficient and scalable.

6. Performance Results

Theoretically, the MPI version would be X times faster than the serial, X being the number of processes used. Of course this result is not in accordance with ours, which can be easily explained, for example, the network load, the cluster computers' specs differences, the algorithm itself, etc.

Due to the processes communication overhead, the MPI version efficiency is easily observed when input matrixes are bigger, which is the reason why we focused our tests on inputs with world size equal or bigger than 100.

World Size	Wolf Breed	Squirrel Breed	Wolf Starv	# of generations	Serial	8 cpus	16 cpus	32 cpus
100	20	50	30	1000	1.24	1.13	1.13	1.03
100	20	50	30	10000	12.58	7.19	9.3	6.74
100	20	50	30	100000	132.31	55.96	74.13	64.29

Table 1 Time in seconds for input matrix 100x100

World Size	Wolf Breed	Squirrel Breed	Wolf Starv	# of generations	Serial	8 cpu	16 cpu	32 cpu
1000	20	50	30	1000	85.61	10.86	7.48	3.9
1000	20	50	30	10000	852.87	134.64	74.41	41.93
1000	20	50	30	100000	8545.3	1093.35	730.29	481.62

Table 2 Time in seconds for input matrix 1000x1000

World Size	Wolf Breed	Squirrel Breed	Wolf Starv	# of generations	Speedup 8	Speedup 16	Speedup 32
100	20	50	30	1000	1.10	1.10	1.20
100	20	50	30	10000	1.75	1.35	1.87
100	20	50	30	100000	2.36	1.78	2.06

Table 3 Speedups for input matrix 100x100

World Size	Wolf Breed	Squirrel Breed	Wolf Starv	# of generations	Speedup 8	Speedup 16	Speedup 32
1000	20	50	30	1000	7.88	11.45	21.95
1000	20	50	30	10000	6.33	11.46	20.34
1000	20	50	30	100000	7.82	11.7	17.74

Table 4 Speedups for input matrix 1000x1000

As we can see on the results above, the speed up isn't nearer the ideal, even though it shows a similar performance to the one predicted theoretically, i.e. the MPI version is more adequate to bigger world sizes.

It's important to mention that the machines where the tests were executed were also used by other students, which may have compromised the times taken, i.e., the presented values may not match the actual performance of our implementation.