Low-Level Parallel Programming:
Assignment 3 – Collision Prevention

Assignment 3 Report

Group 14: Aljaz Kovac, Jakob Nordgren, Simon Jaklovsky

Statement

All group members participated equally on this assignment.

Usage

Run the program like this:

$ demo/demo --implementation OMP file_name

QUESTIONS

Is it possible to parallelize the given code with a simple omp parallel for (compare with Assignment 1)? Explain.

No, that is not possible because the agents are being handled by different threads, which means that there are possible data races that could occur. This would result in agents landing at the same coordinates, which is of course not allowed as the whole point of this assignment is to ensure that collision prevention is always respected despite parallelization.

How would the global lock solution perform with an increasing number of threads? Would the second simple solution perform better in this regard? Can the scenario affect the relative performance of the second simple solution?

The global lock solution would prove to be slower with an increasing number of threads. The second simple solution (we assume here you mean the one where we guard every single location with a separate lock) would perform worse on a bigger scenario as a great amount of locks would have to be produced.

Consider a scenario file designed to generate the worst-case load balancing for your solution. Describe and try to quantify how bad it could get. Worse than a sequential solution?

If all agents are spawned in at the same x-value our code would put them all in the same region to be handled by one thread. We suspect it would perform even worse than the sequential one due to the CAS operations performed at the region boundaries.

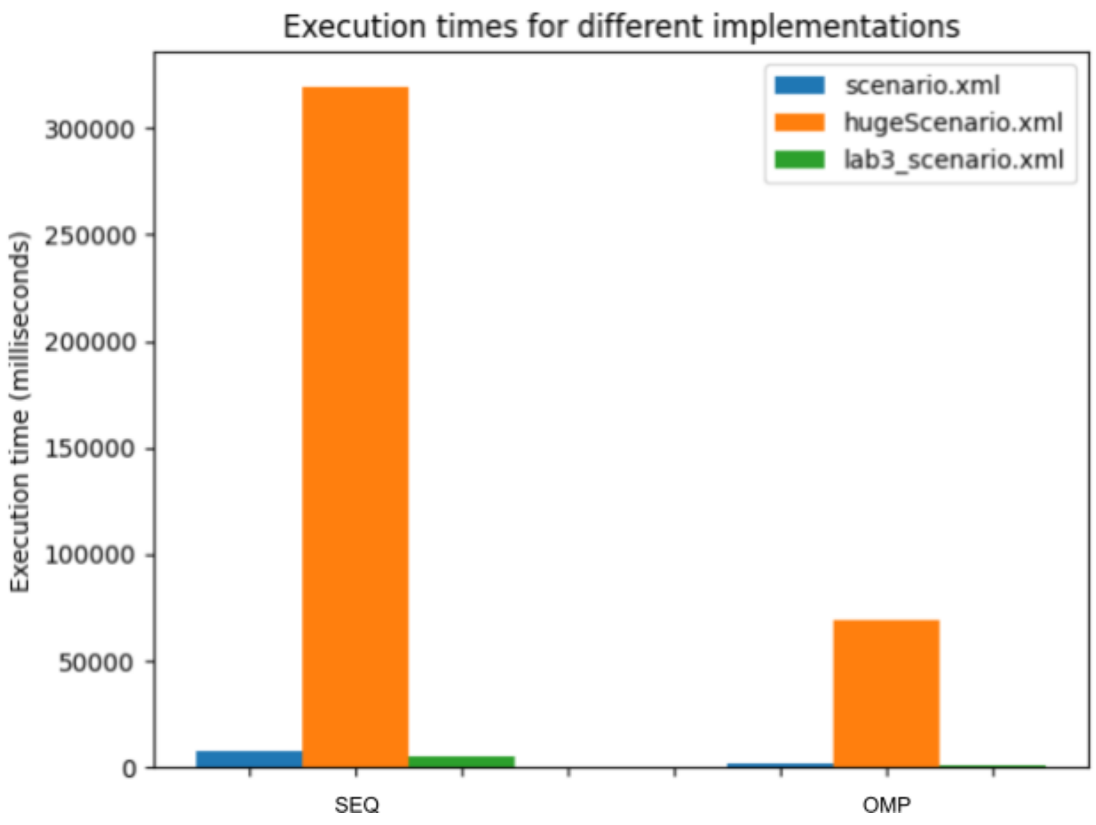
Would your solution scale on a 100+ core machine?

No, not if the amount of agents and size of the domain in which they can move in stays the same. The region splitting is done at the granularity of distinct x-values. On a 100+ core machine with 100+ threads we would not be able to utilize all threads due to the regions not being split at a fine enough level.

TIMING EVALUATION

Timing tests were run on the UPPMAX system, using the following specification:

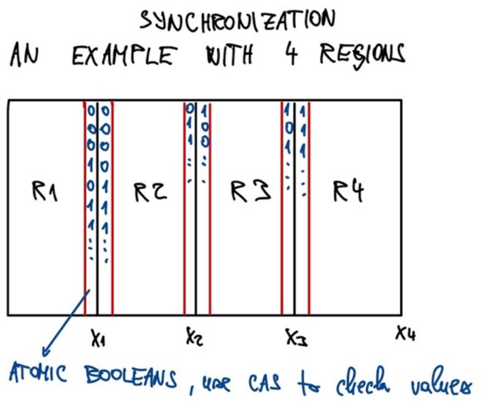
interactive -A uppmax2021-2-28 -M snowy -p core -n 1 -c 8

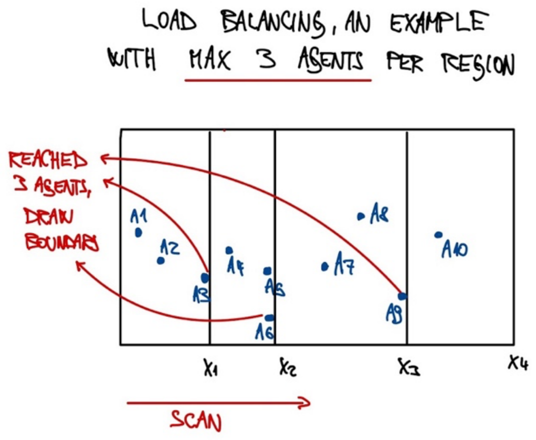


BONUS ASSIGNMENT QUESTIONS

Briefly describe your solution. Focus on how you addressed load balancing and synchronization, and justify your design decisions.

Synchronization: our solution partitions the 2D plane into several regions along the x-axis, cutting them off vertically. Each region is handled by a different thread, and therefore we must ensure that the agents moving from one thread to another are handled correctly. To ensure correct behavior at the borders of regions, we implemented a 2D array which holds all boundaries, and they in turn contain atomic booleans, indicating whether a particular position is taken or not. In the move_atomic() function we then use compare and swap (CAS) to check whether a position at the border is empty or not, and we only move the agent there if it is available.
Load balancing: we sort the vector of agents according to their x-coordinate, and we set the max number of agents we want to allow per region. In this way, all the regions will always be approximately equally busy. We then scan this sorted vector of agents and when we reach the threshold for the max number of agents per region, we take the x-position of the last agent we fit into the region and draw a boundary there. The boundary is then used for synchronization (see above). Please refer to the diagram below for a visual explanation.


SYNCHRONIZATION
AN EXAMPLE WITH 4 REGIONS
R1
R2
R3
R4
X1
X2
X3
X4
ATOMIC BOOLEANS, use CAS to check values


LOAD BALANCING, AN EXAMPLE
WITH MAX 3 AGENTS PER REGION
REACHED 3 AGENTS, DRAW BOUNDARY
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
X1
X2
X3
X4
SCAN

Can your solution be improved? Describe changes you can make, and how it would affect the execution time.

An improvement can be done in the data layout. For each agent in a region we need the positions of its neighbors. In our implementation we are looping through the agents' vector instead of using xArray and yArray. Looping through xArray and yArray is faster due to the data-layout.

Another problem we noticed was that for hugeScenario.xml the majority of agents got stuck in a group unable to reach their destination or move at all. We introduced a back-off to the heuristic if the agent could not move to any of its three alternatives. Introducing randomness to the back-off could perhaps improve the agents' ability to move around and thus reducing the risk of getting stuck.