Agent-based Modeling of Social Phenomena for High Performance Distributed Simulations

Mateusz Paciorek and Wojciech Turek

AGH University of Science and Technology, Krakow, Poland
{mpaciorek,wojciech.turek}@agh.edu.pl

Abstract. Detailed models of numerous groups of social beings, which find applications in broad range of applications, require efficient methods of parallel simulation. Detailed features of particular models strongly influence the complexity of the parallelization problem. In this paper we identify and analyze existing classes of models and possible approaches to their simulation parallelization. We propose a new method for efficient scalability of the most challenging class of models: stochastic, with beings mobility and mutual exclusion of actions. The method is based on a concept of two-stage application of plans, which ensures equivalence of parallel and sequential execution. The method is analyzed in terms of distribution transparency and scalability at HPC-grade hardware. Both weak and strong scalability tests show speedup close to linear with more than 3000 parallel workers.

Keywords: Agent-based modeling · Social models · Scalability · HPC.

1 Introduction

Computer simulation of groups of autonomous social beings, often referred to as Agent-based Modeling and Simulation (ABMS, ABM), is a fast developing domain with impressively broad range of applications. Experts in urban design, architecture, fire safety, traffic management, biology and many other areas use its achievements on a daily basis. Recently, its importance became clearly visible in the face of Covid-19 pandemic, which caused urgent demand for accurate simulations of large and varied populations. Researches continuously work on new modeling methods, models of new problems, general-purpose tools and standards [9], trying to provide useful results. However, reliable simulations of detailed and complex phenomena, observed in the reality by domain experts, require advanced simulation models, large numbers of agents and numerous repetitions. The process of creating, verifying and validating such models is time consuming and requires significant computational effort. The most natural approach is to parallelize the computation and use modern hardware, like GPUs or HPC.

The considered computational problem typically does not belong to "embarrassingly parallel" class. The model update algorithm must repeatedly modify a

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

single large data structure. If the modifications are performed by parallel processes, the access to the data structure must be managed. Importantly, detailed features of the simulation model strongly influence the complexity of the required synchronization. Efficient and transparent distribution of complex ABMs simulation remains an open problem, especially when HPC-grade systems are considered.

In this paper we discuss the existing approaches to the problem of parallel execution of ABMs. We identify several classes of such models in the context of simulation parallelization and synchronization. The conclusions allowed us to propose a scalable method for modeling and simulation of the most challenging of the identified classes, while removing hidden biases in model update order present in existing approaches. The method presented in Section 4 is analyzed in terms of distribution transparency (Section 5) and HPC-grade scalability (Section 6).

2 Distributed Simulations of Agent-based Models

Probably the most significant and successful application of software agents paradigm is related to computer simulation. The methodology of Agent-based Modeling (ABM) [9], is a widely used method for simulating groups of autonomous entities in micro-scale. It generalizes many specific areas, like pedestrian dynamics, traffic simulation, building evacuations and others. The simulation based on ABM has been proven superior to approaches based on differential equation models in many areas. The autonomy and differentiation of particular agents can more accurately reflect phenomena observed in the modeled systems.

Simulation of ABMs is computationally expensive. There are several factors which collectively increase the amount of required computations and memory, including agent and environment model complexity, size of the environment and the number of agents. The desire to simulate larger and more accurate models encourage researchers to investigate the methods for parallelization of ABM simulations, which is the main focus of the presented work. The aim is to improve performance by computing model updates in parallel and collecting the same results as in case of sequential execution. The problem of "transparent distribution" is non-trivial due to the need of updating common model state by parallel processes.

The typical approach to the problem is based on division of the environment model [3]. Parallel process compute updates of agents in the assigned fragments of the model and then perform synchronization of results with the processes responsible for adjacent fragments. This synchronization is the key issue – its complexity strongly depends on model characteristics. In the further analysis we will discuss different types of models and existing synchronization strategies.

The majority of ABM simulations uses discrete representation of the environment as a grid of cells, with well defined neighborhood, distance and possible states of cells. Such approach, which is often imprecisely referred to as cellular automaton (CA), significantly simplifies model data structures and update algorithms, while preserving sufficient expressiveness.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

The problem of parallel update of CA-based simulations, which is discussed in details in [4], is typically solved by defining overlapping margins along the border of the environment fragments, which are available to two processes responsible for updating the fragments. These margins, often referred to as *buffer zones* or *ghost cells*, are updated by one process and sent to its neighbor for reading the state of "remote cells".

Considering parallel state update we should differentiate two classes of update algorithms, which:

1. always update one cell at a time (classic-CA),
2. update two or more cells at a time (CA-inspired).

If a model belongs to the first class, its simulation is an "embarrassingly parallel" problem – mutual exchange of ghost cells solves the model synchronization problem. However, the second class is required by vast majority of ABM simulations. When an agent "moves" from one cell to another, the state of both cells must change atomically and often the result limits possible moves of other agents in the same simulation step. Migration of an agent to a cell shared with a different process might result in a conflict. The conflict can be avoided if the model is deterministic and each process has sufficient information to ensure move safety [10].

Stochastic models prevent predicting updates performed by a remote process, making the problem of model synchronization really hard. Interestingly, classic-CA models do not expose this issue, which has been shown in [11], where a simulation of city development has been successfully parallelized. Also, models which can accept and handle collisions, can be parallelized relatively easy. For example, the simulation of stochastic model of foraminifera (small, yet mobile sea creatures) provides correct results in distributed configuration [3].

The presented analysis shows that many models can be efficiently parallelized with relatively simple synchronization mechanisms. However, the remaining class of models, which are CA-inspired stochastic models with mutual exclusion of actions, is very important in many applications. The problem has been identified in the domains different from ABM, like simulation of chemical reactions [1].

Synchronization mechanisms for distributed update of the considered class of models have been considered in [2]. The authors propose three different strategies to ensure lack of collisions in decisions of particular agents. Two of them (location-ordered and direction-ordered) divide each simulation step into constant number of sub-steps. In the location-ordered strategy only agents distanced by 2 cells are allowed to move simultaneously. In the direction-ordered strategy agents moving in the same direction are allowed to move at the same time. The third strategy, called trial-and-error, requires collecting plans from all agents and solving conflicts using priorities. Agents which were not able to move can try another destination. The procedure is repeated until all agents move or decide to remain in their original cells. All three synchronization mechanisms provide transparent distribution, all three require repetitive communication between processes during a single simulation step. The authors conclude that the location-ordered and direction-ordered methods are more predictable in terms

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

of computational cost, therefore only these methods have been implemented and tested. The location-ordered strategy has been also described in [5] and used for implementing pedestrian dynamics simulation for multiple GPUs.

The features of the described synchronization strategies will be analyzed and discussed in the next section. The conclusions will formulate basis for a new method proposed in this paper.

3 Discussion on Model Update Distribution

The two methods analyzed and implemented in [2], i.e. location-ordered and direction-ordered, ensure constant synchronization costs and do allow to avoid collisions in agent-based simulations. Their usage is however not without consequences to the behaviors that can be observed in the simulation. A mechanism similar to the location-based method has been implemented in the simulation of emergency evacuation in [7] and proved to introduce patterns in the behavior of evacuated people. Depending on the orientation of the identically shaped and sized junction between a staircase and a building floor, a clear prioritization could be observed. In one case the people inside the staircase were consistently blocked by the inflow of the people evacuating the intersecting floor, while in the other case the flow of the people in the staircase could not be interrupted and the floor occupants were forced to wait for the higher floors to fully evacuate.

Fig. 1: Patterns emerging in location-ordered method.

The synthetic example presented in Fig. 1 visualizes the mentioned problem more clearly. In this example the agents try to reach the exit cell marked as EX, at which point they are removed from simulation. The arrows show the direction of movements performed between the previous state and the current one. The first scenario, shown in Fig. 1, leads to the situation where the time steps t_6 and t_7 repeat until all agents present in the top left area of the grid (colored blue)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

are evacuated. No more agents from the bottom area (colored yellow) will be allowed to reach the exit until then. The same is the case for the blue agent at the junction between the two parts of the grid.

This phenomenon will be present in all methods that prioritize some actions above the others and a similar example could be derived for the direction-ordered method. It is important to note that the measurement of the rate of agents reaching the exit cell will not hint at the existence of the problem, as the agents are steadily leaving the grid. Only the observation of more intricate details of the simulation allows to note this behavior, e.g. tracking the distribution of time spent without moving among all the agents. This should not be the case, as the designer of the simulation model will be either unaware of the bias introduced by the method or forced to work around this limitation.

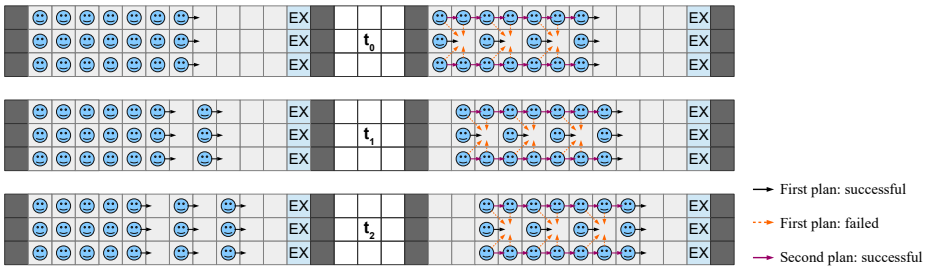

Fig. 2: Patterns emerging in trial-and-error method.

The other method presented in [2], called trial-and-error, introduces the mechanism of collision resolving. Initially, agents are allowed to try to perform an action, but in the case of collision only one of the colliding moves will occur and other agents will be allowed another attempt. This process is repeated until all agents either performed a move successfully or decided to stay in place. The authors dismiss this method as it can generate varying amount of retries that require repeated communications between computational nodes. Taking further look at the implications of this method, one can observe unexpected behavior, which can be seen in Fig. 2. The selective approach to the agents being allowed another move when previous one caused collision generates another pattern. If the circumstances do not allow the agent to move, it will stay in its original cell (Fig 2, left). However, if the agent is allowed to attempt to move, even if it collides, it can then exploit the new movement possibility generated by the previous batch of moves. As shown in the example (Fig. 2, right), majority of agents are allowed to take action that was unavailable to them in the initial state of the grid. As a result, the example presents two configurations of agents that will result almost $1 : 2$ difference in the throughput of the corridor.

This analysis of the existing approaches to the synchronization and collision-handling methods yields several requirements for the approach that could be deemed transparent for the model, while retaining good scalability:

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

- The amount of the communication related to the decision-making and collision handling should be predictable—as suggested in [2].
- No prioritization of moves or agents should be present unless explicitly implemented in the model.
- All agents should have equal capabilities of observing environment, i.e. either all decisions are based only on the initial state or all agents can track intermediate changes in environment and decide accordingly.

These conclusions led us towards a new method of model update distribution. This method, which is the main contribution of this paper, will be described in the next section.

4 Distributed Model Update Algorithm

To satisfy requirements outlined in previous section, a new method has been designed. As only the method of decision-making and conflict-resolving is the main focus of this work, several important assumptions are made to facilitate the understanding the method without unnecessary insights into the other parts of the simulation system:

- Prior to the main body of the simulation the simulation state is divided into parts that are then assigned to their respective "owner" computational nodes.
- The mapping from the identifier of the cell to the owner node is known to each node.
- Although the examples shown in the following sections use the Moore neighborhood, this method can be easily adapted to allow agents to interact with more distant cells.

The crucial concept introduced in the method is a *plan*. One or more *plan* is produced by each agent in each simulations step. Each *plan* consists of the following information:

- *action*—the change the algorithm intends to apply to the state of the handled cell or its neighbor.
- *consequence*—the optional change to the state of handled cell that is required to accompany the application of the action.
- *alternative*—the optional change to the state of handled cell that is required to accompany the rejection of the action.

None of these is guaranteed to be applied to the simulation state. Instead, the proposed distributed simulation update algorithm, after collecting *plans* from all agents, decides which *actions*, *consequences* and *alternatives* are executed.

A good example of this mechanism might be a model in which agents track their age (measured in iterations) and traverse the cells of the grid. The plan of an agent would be following: the action is inserting a copy of the agent with increased age into neighboring cell, the consequence is removing the agent from

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

the original cell, and the alternative is replacing the old agent with its copy with increased age. If a plan is accepted, the result will be the application of action and consequence (add aged agent in neighboring cell, remove from the original cell). If a plan is rejected, the result will be application of alternative (keep aged agent in the original cell).

The possibility of multiple plans might be further explained by extending the example model with reproduction: if agent fulfills some criteria, it is allowed to create a new agent of the same type in adjacent cell. In such case, another planof the same agent could comprise of action of placing a new agent in the target cell, with empty consequence and alternative.

The important assumptions, emerging from this concept, are following:

- No two plans created by the same agent can contradict or depend on each other, as their acceptation or rejection is resolved independently—e.g. the agent cannot try to move into two different cells.
- Consequences and alternatives are not subject to rejection and must always be applicable.

The core of the method is the course of each simulation iteration, which has been described in Algorithm 1. The *exchange* executed in lines 6, 16 and 24 is realized by grouping the exchanged elements by their cell identifiers and sent to the owners of these identifiers.

There are a few model-specific components, marked **bold** in the Algorithm 1:

- A component responsible for the creation of the initial grid state.
- *createPlans* - creation of the plans basing on the the cell and its neighbors.
- *isApplicable* - accepting or rejecting the plan basing on the target cell.
- *apply* - application of the given plan to the given cell.

The actual model part of the implementation is unable to determine whether any given neighbor cell is owned by the current node or is the "ghost cell". The same lack of knowledge pertains to the origin of any plan that is being resolved. Therefore there can be no influence of the distribution of the grid on the decision making algorithm itself.

The only part of the algorithm affected by the distribution and communication between computational nodes is the necessity of the exchange of the plans (Algorithm 1, line 6) if the plan would affect the cell owned by another computational node. The plans are not resolved immediately, which results in all of them being handled in the same point in the iteration. Therefore the only possible effect of the distribution would be the change in the order of their resolving (e.g. locally created plans first, then plans from other nodes in order dictated by the order of communication events). However, this possibility is eliminated by the introduction of the obligatory randomization of the order of plans handling (Algorithm 1, line 7).

As a side note, if the model explicitly requires ordering of the plans application, the randomization step can be easily replaced by sorting (preceded by shuffling, if the order is not total, to ensure no bias).

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

Algorithm 1: Steps of the single iteration of simulation.

```
   // plans creation step
 1 plans ← emptyList;
 2 foreach cell in localCells do
 3 |   cellNeighbors ← getCellNeighbors(cell);
 4 |   plansForCell ← createPlans(cell, cellNeighbors);
 5 |   plans.append(plansForCell);
 6 localPlans ← exchange(plans);
 7 shuffledLocalPlans ← shuffle(localPlans);
   // actions application step
 8 reactions ← emptyList;
 9 foreach plan in shuffledLocalPlans do
10 |   targetCell ← localCells.getCell(plan.getAction().getTargetId());
11 |   if isApplicable(targetCell, plan) then
12 |   |   apply(targetCell, plan.getAction());
13 |   |   reactions.append(plan.getConsequence());
14 |   else
15 |   |   reactions.append(plan.getAlternative());
16 localReactions ← exchange(reactions);
   // reactions application step
17 foreach reaction in localReactions do
18 |   targetCell ← localCells.getCell(reaction.getTargetId());
19 |   apply(targetCell, reaction);
   // "ghost cells" update step
20 edgeCells ← emptyList;
21 foreach cell in localCells do
22 |   if isEdgeCell(cell) then
23 |   |   edgeCells.append(cell);
24 ghostCells ← exchange(edgeCells);
```

5 Transparency of Simulation Distribution

The method described in previous section has been added to the framework discussed in detail in [3]. To ensure the method does fulfill the transparency of distribution requirements presented in Section 3, the experimental verification was designed and executed.

The exemplary model implemented using the new method was a simple predator-prey scenario involving rabbits and lettuce, similar to the one described in [3]. Each cell is either occupied by a lettuce agent, a rabbit agent, or is an empty cell. In each iteration each agent can perform an action. Lettuce is allowed to grow by creating new lettuce agent in a random neighboring cell, limited by the growing interval. Rabbit is allowed to take one of the three actions, depending on the energy parameter of the agent: if below zero, the rabbit dies (is

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

removed from the grid); if above reproduction threshold, the rabbit reproduces by creating a new rabbit agent in a random neighboring cell; otherwise, the rabbit moves towards the lettuce, expending some of its energy.

In addition to the energy of the rabbit agent, both agent types track their "lifespan", i.e. the number of iterations since their creation. No two agents of the same type can occupy the same cell. If any action leads to the situation where agent of one type would be in the same cell as the agent of the other type, the lettuce is consumed (removed from the grid) and the rabbit increases its energy level. The reproduction threshold, energy gained from consumption, energy cost of movement, initial energy of rabbits and lettuce growing interval are the main parameters of the simulation and can be adjusted.

The implemented system collects metrics during the simulation execution. In the case of the mentioned simulation, the metrics collected for each iteration are following (naming uses camel case notation mirroring the one used in the implementation):

- rabbitCount - the number of the rabbits present on the grid.
- rabbitReproductionsCount - the number of rabbit reproductions.
- rabbitTotalEnergy - the total energy of all rabbits present on the grid.
- rabbitDeaths - the number of rabbit deaths.
- rabbitTotalLifespan - the cumulative lifespan of rabbits that died.
- lettuceCount - the number of the lettuces present on the grid.
- consumedLettuceCount - the number of the lettuces consumed.
- lettuceTotalLifespan - the cumulative lifespan of consumed lettuces.

The HPC system used to run the experiments was a Prometheus supercomputer located in the AGH Cyfronet computing center in Krakow, Poland. According to the TOP500 list, as of November 2020 it is 324th fastest supercomputer. Prometheus is a peta-scale (2.4 PFlops) cluster utilizing HP Apollo 8000 nodes with Xeon E5-2680v3 CPUs working at 2.5GHz. Each of the nodes connected via InfiniBand FDR network has 24 physical cores (53,604 cores total).

5.1 Stochastic experiments

The first set of experiments was performed within 1000 iterations each and using the constant size of the grid: 480x500 cells. To observe the influence of the changes in parameters on the simulation course and results, four batches of experiments were conducted, each using different set of parameters, later referred to as "variants" numbered 0–3. The detailed description of the parameters and the variants can be found in [8] (where the "variant 0" is referred to as "default"). Each variant was then executed in different degrees of distribution - using 1, 2, 4, 6, 8, 10, 20, 40, 60, 80 and 100 nodes, 24 cores each. Each unique combination of the variant and the distribution degree was executed 10 times to obtain sample of size necessary for the variance analysis used in later steps.

It is clearly visible that the simulation model displays high variance in the results, even within the identical configurations. As is visible in Fig. 3, which

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

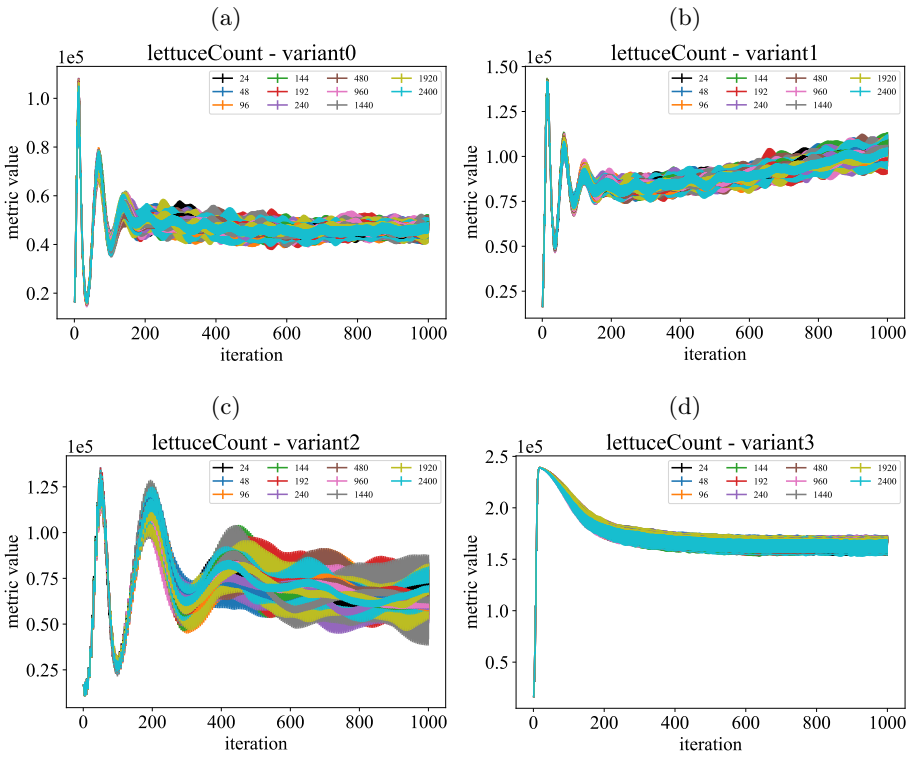

Fig. 3: Comparison of lettuceCount metric over time in stochastic execution. Each plot represents different model parameters configuration. Series correspond to means with standard deviations collected for different distribution degree.

Table 1: Summary of average p-values and percentage of p-values below threshold

	variant 0		variant 1		variant 2		variant 3	
	avg. p	<0.05	avg. p	<0.05	avg. p	<0.05	avg. p	<0.05
rabbitCount	0.47	4.2%	0.42	3.9%	0.48	1.8%	0.44	1.4%
rabbitReproductionsCount	0.47	2.9%	0.44	4.2%	0.48	2.7%	0.45	1.1%
rabbitTotalEnergy	0.47	3.9%	0.42	4.1%	0.48	2.5%	0.44	1.4%
rabbitDeaths	0.48	3.0%	0.44	4.7%	0.48	2.7%	0.45	2.7%
rabbitTotalLifespan	0.48	3.3%	0.44	3.9%	0.48	3.0%	0.46	3.2%
lettuceCount	0.55	2.4%	0.44	5.7%	0.52	0.1%	0.47	1.8%
consumedLettuceCount	0.47	2.8%	0.42	4.5%	0.48	3.0%	0.44	1.6%
lettuceTotalLifespan	0.50	3.2%	0.45	7.3%	0.47	5.2%	0.53	1.5%

represents the change of one of the metrics (*lettuceCount* - chosen randomly) in the time, the bands created by the means are noticeably wide and do not overlap perfectly. However, each of the variants presents some tendency obeyed by all

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

series representing different degrees of distribution. Both Fig. 3a and 3b display intense oscillations at the start, followed by the stabilization of the former and rise of the latter. Fig. 3c shows slower oscillations and very high variance in the following part, and Fig. 3d shows sharp rise at the beginning and a slow decrease to a nearly constant value.

To refrain from the imprecise visual analysis, the Kruskal-Wallis test [6] was used to determine whether the values obtained from different numbers of nodes are likely to represent the same distribution. For each iteration of each variant, the 10 collected series of each metric were treated as a sample for the respective distribution degree. For the majority of the iterations the values exceeded the threshold, with sporadic segments where the p-value is temporarily below the threshold.

To further evaluate the possibility of the samples not representing the same distributions, the average p-value for each plot was calculated, with the addition of the percentage of the values below the threshold. The results including all the metrics are summarized in Table 1. As the average p-value was always significantly above the threshold ($0.42 - 0.55$) and the percentage of negative results was low, the most plausible explanation for the occurrences of the negatives is the high variability of the model itself. Possibly the more extensive experiments (e.g. sample sizes larger than 10) would eliminate the outlying results.

5.2 Deterministic experiments

To ensure that the minor discrepancies observed in the previous experiments are a result of the high sensibility of the model to the random decisions, another set of experiments was conducted. The general course and parameters of the experiment remained identical to the previous one, but the random factors were completely eliminated from the model and the method. Randomness in the moves of agents was replaced by the direction prioritization, while the randomized traversal of the plans was substituted with sorting. Instead of 10 times, each instance was executed only once, as there is no explicit randomness present in the system.

In each instance all collected data differed only at the least significant decimal digits due to differences in rounding, as the metrics are collected and saved separately by each core. This behavior is a result of the aggregation and formatting of the metrics, and has no effect on the course of simulation. No other differences were present. The comparison footage[1] of sample 100x100 cells experiment executed on single core and on 100 cores (10x10 cores, 10x10 cells per core) is available for additional visual verification of distribution transparency.

6 Scalability Experiments

As the crucial role of the implemented method is related to the handling of the distribution and the coordination of the communication between computational

[1] https://youtu.be/9W-zmyQo-K8

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

nodes, the scalability tests were necessary. Tests were conducted using the same implementation and model as described in previous section.

6.1 Weak scaling

The first scalability experiments were the weak scaling tests. The size of the problem was variable to achieve constant size per core—1e4 cells per core. The experiment was executed 5 times on each of the following numbers of nodes: 1, 2, 4, 6, 8, 10, 20, 40, 60, 80, 100 and 120, which translates to the number of cores ranging from 24 to 3600. Therefore, the final sizes of the grids varied from 2.4e5 cells to 3.6e7 cells. Each run consisted of 1000 iterations, the first 100 of which were omitted in final calculations to exclude any outlying results caused by the initialization of the simulation system. The execution times measured on each core were averaged to achieve a single time value for each run.

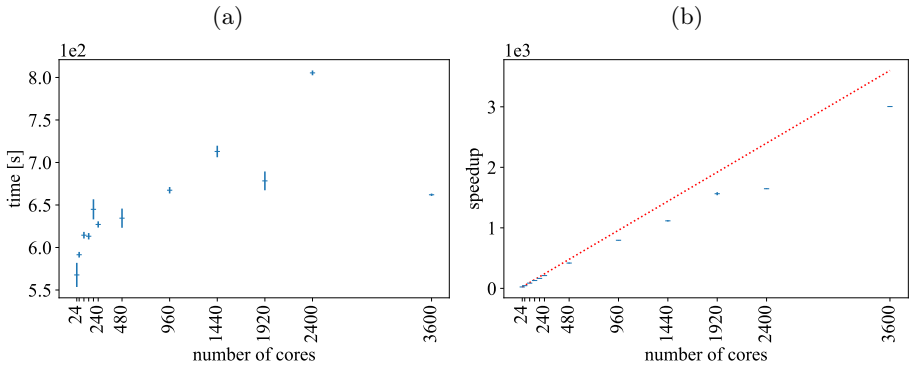

Fig. 4: Execution times and speedup for weak scaling.

The results were grouped by the number of cores and aggregated into means and standard deviations, which are presented in Fig. 4. The first plot (Fig. 4a) presents the execution times of the iterations 100-1000 for each number of cores. The vertical lines show the standard deviation of the measurements. Additionally, the speedup adjusted to weak scaling experiments (i.e. with corrections for the changing problem size) was calculated. The average execution time on 1 node was used as the reference time. This metric is shown in Fig. 4b, with the ideal linear speedup marked with red dotted line. After the initial nearly-perfect scalability, the values became lower than the ideal. However, the deviation from the ideal was expected due to the non-negligible amounts of communication.

6.2 Strong scaling

The second scalability experiments were the strong scaling tests. The problem size was kept constant—2.4e7 cells. Due to the memory limitations, the lower

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

numbers of nodes were not included, and the experiment was executed on 10, 20, 40, 60, 80, 100, 150 and 200 nodes, that is 240 - 4800 cores. The effective problem size for each core ranged from 1e5 cells to 5e3 cells. All other aspects—number of iterations, data collection and aggregation—were identical to the weak scaling experiments.

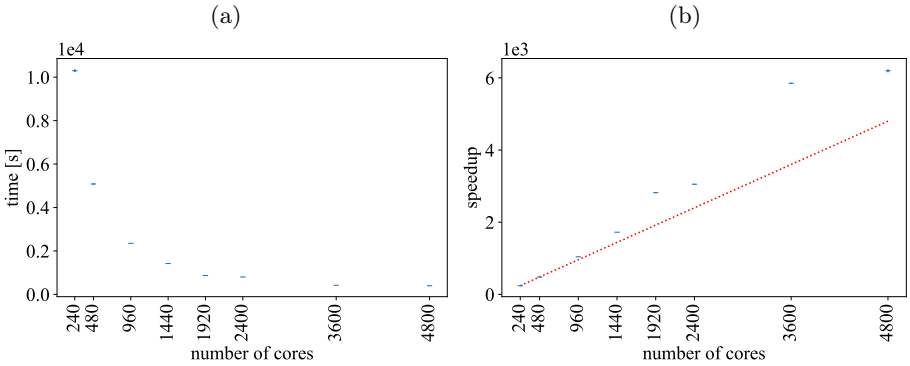

Fig. 5: Execution times and speedup for strong scaling.

As shown in Fig. 5a, the measured execution times follow the roughly hyperbolic shape, which is expected for this type of experiment. The speedup was calculated with the reference value of the average execution time on 10 nodes (240 cores), as the size of the problem did not allow it to be computed in more coarse distribution. The speedup results (Fig. 5b) suggest superlinear scaling, as the values for the larger numbers of cores are above the ideal (red dotted line). The explanation for such results can be the alleviation of the pressure on the memory, which were the reason the experiments with less than 10 nodes were not feasible. As the system is running on JVM, the memory management can introduce additional overhead when the problem parts are large in comparison to the available memory.

7 Conclusions and Further Research

In this work we analyzed the issues present in popular approaches to the problem of ABM simulation distribution. The most important ones concerned the introduction of undesired—and, in some cases, difficult to detect—changes to the simulation model. We proposed the new approach to this problem that ensures no changes to the model, unless the simulation designer intends to introduce them. This solution proved to retain the high scalability expected from the efficient distribution method, while displaying no influence on the simulation outcomes, regardless of the degree of distribution. The method has an additional advantage

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32

of flexibility of the neighborhood definition - the approaches described in Section 3 would require more synchronization cycles to achieve the same.

To further explore the new solution, we intend to focus on adapting models based on real life scenarios. Our concurrent work suggests that it is possible to use this method in the microscopic pedestrian dynamics simulation of epidemic spread in urban environment. Another direction of study possible due to the qualities of the proposed solution is the adaptation of models that require interaction with non-adjacent cells (outside of Moore neighborhood).

Acknowledgements The research presented in this paper was partially supported by the funds of Polish Ministry of Science and Higher Education assigned to AGH University of Science and Technology. This research was supported in part by PLGrid Infrastructure.

References

1. Bezbradica, M., Crane, M., Ruskin, H.J.: Parallelisation strategies for large scale cellular automata frameworks in pharmaceutical modelling. In: 2012 Int. Conf. on High Performance Computing & Simulation (HPCS). pp. 223–230. IEEE (2012)
2. Bowzer, C., Phan, B., Cohen, K., Fukuda, M.: Collision-free agent migration in spatial simulation. In: Proceedings of 11th Joint Agent-oriented Workshops in Synergy (JAWS 2017), Prague, Czech (2017)
3. Bujas, J., Dworak, D., Turek, W., Byrski, A.: High-performance computing framework with desynchronized information propagation for large-scale simulations. Journal of Computational Science **32**, 70–86 (2019)
4. Giordano, A., De Rango, A., D'Ambrosio, D., Rongo, R., Spataro, W.: Strategies for parallel execution of cellular automata in distributed memory architectures. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). pp. 406–413 (2019)
5. Kłusek, A., Topa, P., Wąs, J., Lubaś, R.: An implementation of the social distances model using multi-gpu systems. The International Journal of High Performance Computing Applications **32**(4), 482–495 (2018)
6. Kruskal, W.H., Wallis, W.A.: Use of ranks in one-criterion variance analysis. Journal of the American statistical Association **47**(260), 583–621 (1952)
7. Paciorek, M., Bogacz, A., Turek, W.: Scalable signal-based simulation of autonomous beings in complex environments. In: International Conference on Computational Science. pp. 144–157. Springer (2020)
8. Paciorek, M., Bujas, J., Dworak, D., Turek, W., Byrski, A.: Validation of signal propagation modeling for highly scalable simulations. Concurrency and Computation: Practice and Experience p. e5718 (2020)
9. Railsback, S.F., Grimm, V.: Agent-based and individual-based modeling: a practical introduction. Princeton university press (2019)
10. Turek, W.: Erlang-based desynchronized urban traffic simulation for high-performance computing systems. Future Generation Computer Systems **79**, 645–652 (2018)
11. Xia, C., Wang, H., Zhang, A., Zhang, W.: A high-performance cellular automata model for urban simulation based on vectorization and parallel computing technology. International J. of Geographical Information Science **32**(2), 399–424 (2018)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_32