Computer Science • 21(3) 2020 https://doi.org/10.7494/csci.2020.21.3.3772

Mateusz Starzec
Grażyna Starzec
Mateusz Paciorek

DESYNCHRONIZATION OF SIMULATION AND OPTIMIZATION ALGORITHMS IN HPC ENVIRONMENT

Abstract *The need for the scalability of an algorithm is essential when one wants to utilize an HPC infrastructure in an efficient and reasonable way. In such infrastructures, synchronization affects the efficiency of the parallel algorithms. However, one can consider introducing certain means of desynchronization in order to increase the scalability. Allowing certain messages to be omitted or delayed can be easily accepted in the case of metaheuristics. Furthermore, some simulations can also follow this pattern and thereby handle bigger environments. The paper presents a short survey on the desynchronization idea, pointing out already obtained results, or sketching out future work focused on scaling the parallel and distributed computing or simulation algorithms.*

Keywords scalability, desynchronization, simulations, optimization algorithms

Citation Computer Science 21(3) 2020: 319–333

1. Introduction

Since the accessibility of clusters or even HPC environments is becoming more and more common, the scalability of the algorithms gains importance. This scalability enables algorithms to solve huge tasks faster and simulate bigger problems.

A lot of work has been done to scale algorithms on multi-processor computers where the issue was the slow access to the system memory, which constrains the efficient utilization of multiple processors. An alternative way of achieving a speedup on a single computer is to use the GPUs in order to perform fast matrix computations; however, this requires a specific algorithm construction and data transfer between the system and the GPU memory. Working on the scalability across the cluster encounters a different obstacle – costly communication over the network between nodes enters into the equation.

As the communication between nodes may easily reduce the whole system's performance, there is a need to reduce its amount and organize it in such a way that impacts the computational efficiency as little as possible. It is rarely possible to ensure an identical load on all computational nodes, especially in a heterogeneous environment. Inequalities in the execution time of a single algorithm step will cause the fastest nodes to idly await the completion of the step on the slowest nodes, effectively wasting potential computational capabilities. The concept of desynchronizing inter-node data exchange aims to remove the synchronization points from the algorithm and replace them with an asynchronous data exchange. This might lead to the modified behavior of the algorithm when compared to the non-desynchronized approach.

In this paper, we review several methods of applying desynchronization to inter-node communication in various optimization algorithms and simulations. We note some issues that one should be aware of when applying such desynchronization to computations. The modifications enable high scalability but might also lead to unsatisfactory or even invalid results under some circumstances. The majority of the discussed cases are referenced from previous works related to the topic discussed. However, we decided it would be helpful to bridge the gap between optimization algorithms and complex simulations by introducing a new example – a simple implementation of Conway's Game of Life [4].

It is to note that the main aim of this paper is to present a short survey of the findings connected with the application of desynchronization in metaheuristic computing, also pointing out the possibility of applying the same mechanism (though more planning is required) to simulations, thus supporting the capability to efficiently scale the concurrent versions of the algorithms under consideration on parallel and distributed HPC infrastructures.

In Section 2, we sum up the current research status regarding the scalability of algorithms. Section 3 describes the concept of desynchronization in an abstract way and proposes some example applications for simulations and optimization algorithms. The next chapter presents example experiments regarding the desirable and unwanted results of desynchronization. In Section 5, we sum up the whole article.

2. Scalability in computations

As computation clusters and HPC environments become more accessible, the horizontal scalability of algorithms is becoming more and more important. A lot of articles have been published regarding the scalability of both optimization algorithms [17,18] and simulations [2,13,14]. However, the majority of this research focuses on multi-processor single-node scalability [1, 7] or, at most, dozens of computational nodes [8, 11]. The synchronization points related to updates of global knowledge are omnipresent. A lack of strict synchronization is treated as a risk of data loss, which leads to invalid results.

The optimization algorithms based on a population of agents are relatively easy to model and implement in parallel. As an example, particle swarm optimization [9] iteratively improves a set of candidate solutions basing on the best solutions collected in the previous iterations. Each solution can be handled independently, so it is straightforward to run in parallel. The algorithm was proven to be usable in inverse rendering with good scalability for up to ten nodes. Unfortunately, the efficiency drops for more computational nodes, and the tests were conducted for only up to 30 nodes [11]. The parallel implementation of PSO shown even super-linear speedups; however, the tests utilized only a single multi-core computer [1].

Ant colony optimization [5] runs multiple independent agents constructing candidate solutions based on a pheromone matrix that is updated after each iteration. The solution-construction processes are independent, so they could be run in parallel. The parallel version of the algorithm was adapted to a waste collection vehicle routing problem, reaching 6.8 speedup on a 24-thread processor [7]. Ilie and Bădică et al. [8] presented an agent-based approach to ACO modeling. This article reports very good scalability for up to 7 nodes on the *gr666* problem from TSPLIB[1]. The algorithm reaches solutions comparable to the standard ant system. Unfortunately, no results were presented for large-scale TSP problems nor for larger clusters.

As we have shown in [18], ACO can reach good scalability for up to hundreds of nodes in an HPC environment. The proposed architecture keeps the pheromone matrix parts distributed across all nodes and uses desynchronized updates in order to achieve good scalability without a noticeable deterioration of the result quality. The good scalability allows for increasing the size of a colony, which improves the achieved results [16, 17]. In [17], we have proposed many ways to efficiently employ large ant colonies in order to outperform algorithms like MMAS in terms of solution quality. What is more, the higher number of ants enables the faster optimization of big problem instances.

The distributed traffic simulation system [14] was proven to be able to efficiently utilize an HPC environment with 800 computational nodes. In the simulation, the traffic network was distributed across nodes with a limited desynchronization, allow-

[1] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

ing the processes to compute a few iterations without waiting for updates from the neighbor crossings. The simulation achieved super-linear scalability.

Another approach to distribution and desynchronization was shown in [2]. Several simulation models were adapted to use the signal propagation method, which was designed to minimize the required communication between computational models. The resulting framework has proven to be highly scalable, efficiently utilizing thousands of computational cores.

The idea of desynchronization was also utilized in our publications regarding the iterated prisoner's dilemma with a large population of agents [13,15]. The experiments showed that desynchronization does not affect the behavior of a big population in any noticeable way. Moreover, the system achieved much better scalability than the version with a standard (synchronized) communication model between nodes, especially for environments with dense neighborhoods.

Desynchronization is a way to improve the scalability of the algorithms; however, it requires foresight and awareness of its potential impact on the algorithm's behavior. In the next section, we present a discussion on the desynchronization of metaheuristic and simulation algorithms.

3. Desynchronization in computations

One of the major issues regarding the scalability of algorithms is the access to global information. In a large cluster, the global knowledge may be stored on a master node and updated in a synchronous manner after each algorithm step. It takes some time to transfer the update data (any data collected from the nodes used for updating the global information), and the cluster is blocked from running another part of the computations until the synchronization is done, leading to the performance regression and lower scalability.

The basis of the desynchronization idea is to allow the algorithm to run when the global knowledge update is still in progress. The algorithm should start another iteration based on the old data and switch to the updated one as soon as the update is done, even in the middle of the iteration. The computations should be parallel to the updates of global knowledge in order to use the cluster resources more efficiently. Depending on the algorithm type, the parallelism may lead to a loss of result accuracy and conflicts in the global data updates, so each application should be considered carefully and individually.

One of the possible application fields are optimization algorithms, where a small delay of the global knowledge update will not be critical but the improvement of scalability will be noticeable. The negative influence of non-deterministic data losses or conflicts may be compensated by the possibilities facilitated by the high scalability.

On the other hand, the desynchronization of simulations is a substantially different problem. One of the significant differences between simulations and optimization algorithms is that a simulation represents a scenario in which intermediate steps might

be as important as the final result. This is significantly different from the optimization problems where only the final result is important in most cases. Additionally, the simulation state is often more complex and more susceptible to small changes. Therefore, desynchronization might not only cause simulations to yield imprecise results but also invalidate all observations made about the behavior of the system.

Fortunately, not all simulations employ deterministic mechanisms. In the case of a simulation algorithm that relies heavily on randomness, small discrepancies caused by the desynchronization may be less significant as compared to the variance of the results expected to occur naturally. Moreover, the more fine-grained the simulation is, the lesser the impact is of a single error or conflict to the whole process.

3.1. Desynchronization in optimization algorithms

In the case of optimization algorithms, we may often assume that small inaccuracies in global information do not result in significant changes to the final results. This section describes the application of desynchronization to various optimization algorithms.

The particle swarm optimization algorithm [9] consists of multiple independent agents – particles. It can be easily distributed to run on a cluster where each node can handle computations of some particles. The algorithm needs to know the global best solution in order to find new solutions. In the standard implementation, the global best solution should be updated after each iteration, which can reduce the system's scalability. The desynchronized algorithm could keep the global best solution locally and broadcast an update to all other nodes when it finds a better one. The particles could move independently based on the current local value of the global best solution. This might introduce some delays in knowledge distribution but should not affect the algorithm's efficacy, as the knowledge will be eventually consistent.

Social cognitive optimization [19] is another meta-heuristic optimization algorithm that uses global knowledge based on the data accumulated by independent agents. This creates a solution set (social sharing library) and updates it after each iteration with the solutions created by all agents in the system. The agents could be easily distributed in the cluster environment; however, updating the social sharing library requires the synchronization of all processes in the standard implementation. The desynchronized version may allow the agents to build the next solutions based on the current content of the sharing library and apply updates asynchronously. The sharing library may be maintained independently on each node, and the proposed solutions may be broadcasted to all nodes if they are good enough to replace one of the solutions currently contained in the list. The process of library update is not deterministic, so the content of the libraries may differ between nodes. Figure 1 presents an example of the communication sequence; it shows that the updates from the remote nodes are independent from the agents' iterations.

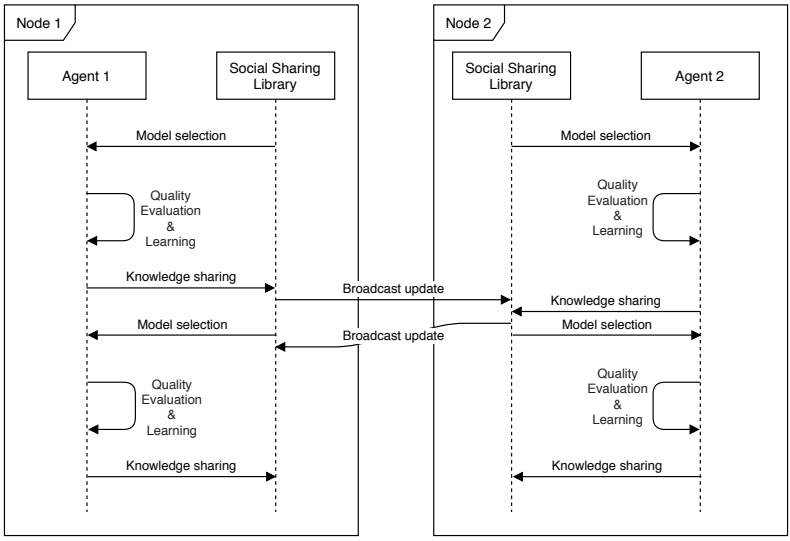

Figure 1. Social cognitive optimization – desynchronized social sharing library update

Evolutionary multi-agent systems [3] use the idea of islands as separated computation environments and utilize migrations as a communication and knowledge-exchange mechanism. The islands are feasible to be distributed across the computation nodes, and the desynchronized implementation should not synchronize the islands after each iteration in order to perform migrations of the solutions. The migrations may be handled asynchronously to increase the scalability of the system. Figure 2 presents an example of overlapping iterations and migrations.

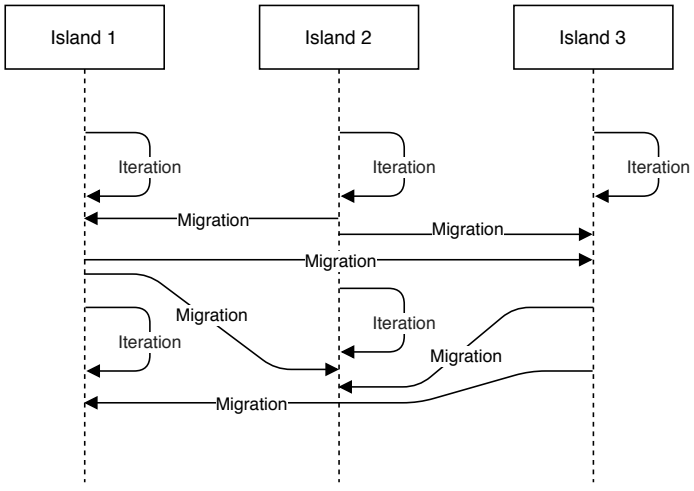

Figure 2. Evolutionary multi-agent systems – desynchronized migration process

After each iteration, each node can send an asynchronous message with the solutions to be migrated and handle such incoming messages. Sending and receiving the messages is not synchronized and may overlap with the iterations – this also means that the size of the population on an island can vary.

3.2. Desynchronization in simulations

In the case of simulations, discrepancies may have a significant impact on the final results. In this section, we describe how the desynchronization ideas may be applied to examples of simulations.

Conway's Game of Life [4] uses a grid as the simulation environment. In the case of a huge simulation, the grid could be distributed across the computation nodes; however, this requires inter-node communication on the edges of the grid's parts during each iteration. The synchronous data exchange may reduce performance, so it could be done asynchronously. On the other hand, the asynchronous approach requires caching the state of the remote neighbors, which could lead to altering the simulation results. Figure 3 presents an example of the grid distribution, with the cached rows represented as gray. The arrows represent the migration of the statuses after each iteration.

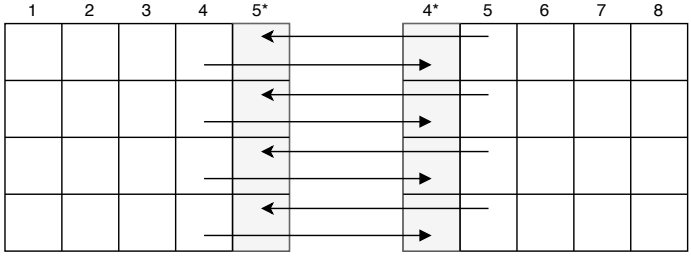

Figure 3. Conway's Game of Life – grid distribution with data caching and updates

A simulation of social processes (like the iterated prisoner's dilemma or a gift-exchange game) in a big society can be organized with some kind of neighborhood; e.g., a two-dimensional grid. To improve the scalability of the simulation, the neighborhood should be distributed in such a way as to minimize inter-node pairs. In order to further reduce the inter-node communication, the remote agents could be represented by local copies, which could asynchronously exchange their status updates with the original agents. The local copy should behave like the original one and provide it with updates regarding the results of the moves it made. Figure 4 presents the idea of a local representation of the remote agent. The communication between the nodes is replaced by the communication with local copies of the remote agents and the periodic synchronization with the original agents. This will reduce the network's communication and improve scalability.

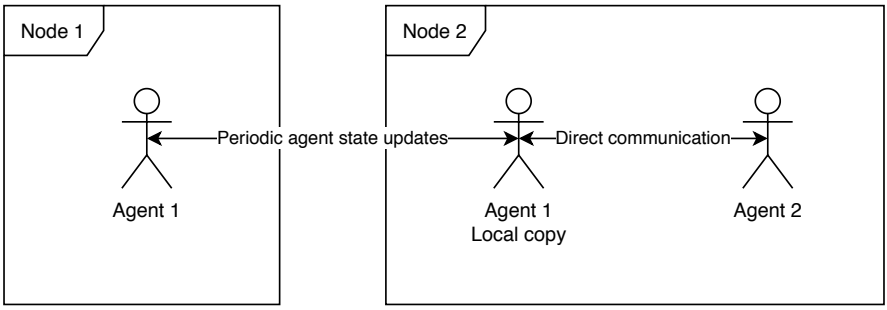

Figure 4. Social simulations – local copy of remote agents

More-complex simulations introduce much more space for the mistakes to either diminish or amplify. The main focus of [12] was to analyze the impact of desynchronization applied to several simulations: the simple predator-prey scenario, fire emergency evacuation, and microscopic organisms development on a seabed. The simulations represented a variety of levels of environment complexity and determinism. Each simulation was desynchronized in a similar manner – the grid was divided into parts and distributed among the computational nodes. The nodes were operating on their parts of the grid and periodically sending updates to the neighboring nodes. As a result, the algorithm had to be capable of resolving conflicts arising from the incoming data potentially contradicting the decisions made locally.

4. Experimental results

The implications of desynchronization may be different for various applications. In this section, we present the experimental results for some optimization algorithms and simulations.

4.1. Ant colony optimization

In the previous publication [18], we described and tested the desynchronized version of ant colony optimization. The idea of the algorithm was to remove the synchronization point at the end of each iteration (when the ants update the pheromone matrix). Instead of synchronizing the whole cluster, each ant submits its solution into a batch and immediately starts to create another solution (without waiting for the update of the pheromone matrix).

The main findings of the experiments conducted for the aforementioned publication are as follows. The algorithm reaches 76% efficiency on 400 computation nodes (see Tab. 1, $Efficiency = Speedup/Nodes$). The scalability enables calculations based on huge colonies, which leads to higher exploration and better final results. Table 2 shows that the bigger colony finds better final results when the exploration is utilized efficiently. The advantages of high scalability outweigh the drawbacks of desynchronized pheromone matrix updates.

Table 1
Desynchronized ACO speedup from [18]

Nodes	2	5	10	50	100	200	300	400
Distributed ACO	2.13	6.22	10.33	43.31	62.04	–	–	–
Desynchronized ACO	1.89	5.38	10.75	46.71	86.64	161.67	235.41	304.58

Table 2
Desynchronized ACO final results compared with MMAS from [18].

	Result
Best-known solution	378,032
MMAS (25 ants)	$494,735.2 \pm 2,983.4$
MMAS (250 ants)	$489,201.8 \pm 1,481.6$
Desynchronized Ant System (250 ants)	$463,671 \pm 10,727.0$
Desynchronized Ant System (500 ants)	$459,025 \pm 10,002.5$
Desynchronized Ant System (2500 ants)	$447,427 \pm 3,153.9$
Desynchronized Ant System (10k ants)	$443,377 \pm 6,392.4$

4.2. Iterated prisoner's dilemma

We have also applied the desynchronization to a simulation of the iterated prisoner's dilemma with a big society [13]. All agents are distributed across the computation nodes. When the game requires communication with a remote agent, it communicates with its local representation, which periodically synchronizes its state with the original agent. Figure 5 presents the results of the experiment from [13].

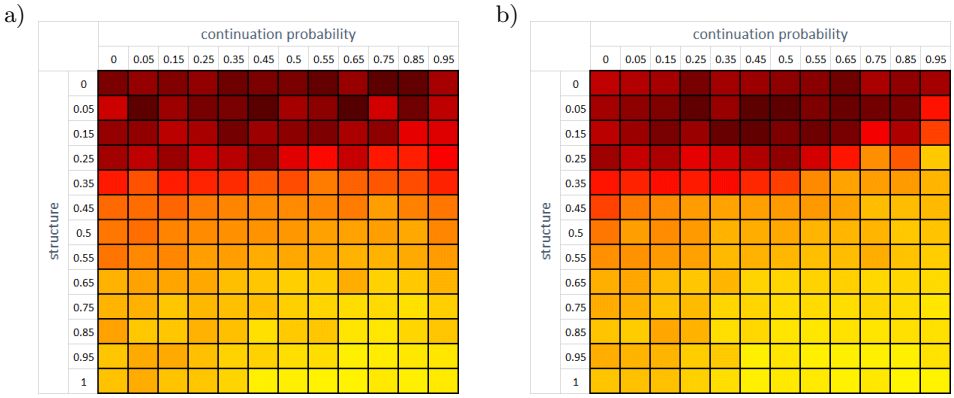

Figure 5. Desynchronization influence on IPD simulation from [13]:
a) without desynchronization; b) with desynchronization

It shows the level of cooperativity in the society depending on two parameters: the probability of playing the next turn with the same opponent (continuation probability), and the probability that an agent will be paired with another with a similar

strategy (structure). The dark color means low cooperativity, and bright yellow means a high level of cooperation among agents.

The desynchronization enabled us to simulate much bigger societies; at the same time, the results imply that this did not significantly influence the simulation results.

4.3. Conway's game of life

One of the most important problems described in the previous chapter was the impact of desynchronization on the simulation result. As an extension of the survey, we decided to further explore this issue using Conway's Game of Life [4] as a simple and well-recognized example. The simulation was implemented in a single process and emulated some conditions that are possible in a desynchronized environment (such as latency or packet losses).

In the simulation, we used a board with 10 rows and 20 columns. In each iteration:

- any black cell with two or three black neighbors remains black,
- any gray cell with three black neighbors becomes black,
- all other black cells become gray, and all other gray cells remain gray.

We assume that there are no neighbors outside the board. The consecutive board states in the standard simulation are presented in Figure 6a.

In order to simulate the environment with two nodes, we split the board into two parts (as presented in Figure 3). Figure 6b presents the results with the cache updates delayed by one iteration (simulating latency in inter-node communication). The differences in the results appear in the center of the board and spread with each iteration.

Figure 6c shows the results of the simulation that drops every fourth cache update (simulating packet losses in inter-node communication). The result after 13 iterations also differs from the standard simulation. The interruption is not as severe as in the previous example, so the differences are smaller (but still noticeable).

The results prove that desynchronization is not suitable for applications where deterministic and consistent results are critical. On the other hand, when an algorithm is randomized, delays in the cache updates may be considered to be another random factor, which is not significant in terms of interpreting the results.

4.4. Complex simulations

The results obtained in [12] varied significantly among the simulation models. Two of the simulations – fire emergency evacuation and microorganism (foraminifera) habitat – were successfully distributed and desynchronized. The influence of the distribution proved to be marginal as compared to the changes caused by modifying the simulation parameters. An example of the results presented in [12] can be seen in Figure 7: each group of points marked on the x-axis represents one set of input parameters, while each of the series represents the degree of distribution; i.e., the number of parts into which the grid was divided.

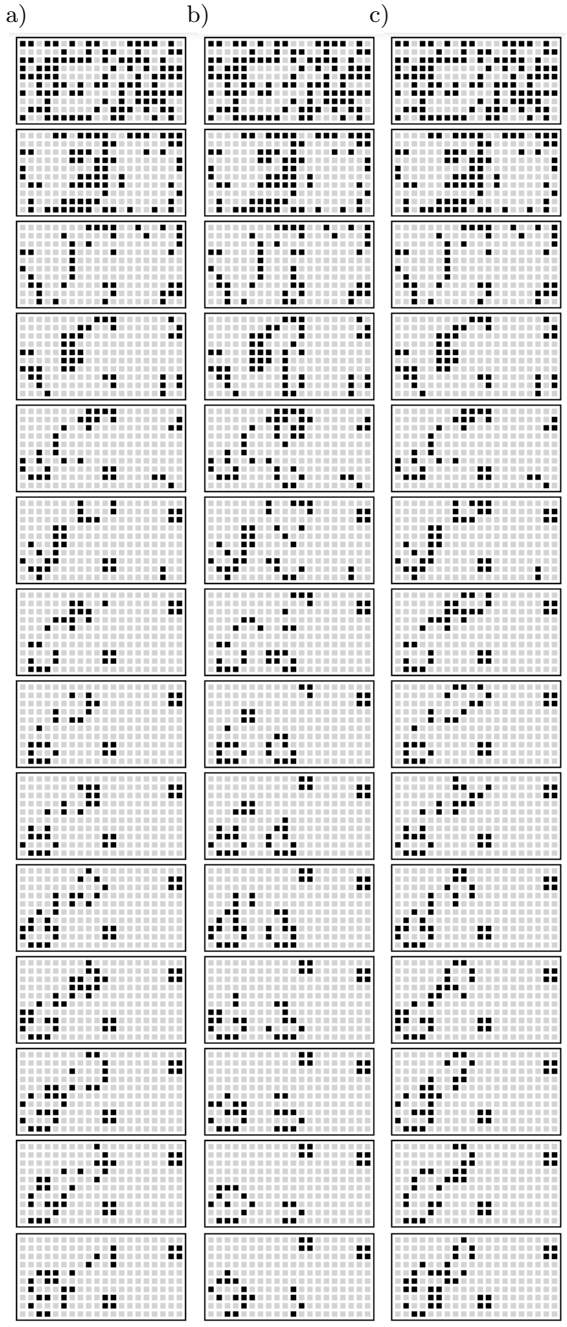

Figure 6. Conway's Game of Life simulation with two nodes:
a) simulation without delays; b) simulation with cache update delayed by one iteration;
c) simulation with every fourth cache update lost

Distribution had some impact on the measured values, which was the total energy among all simulated organisms in this case; however, more important was that the differences among configuration variants were noticeably greater than the differences between the values in each group.

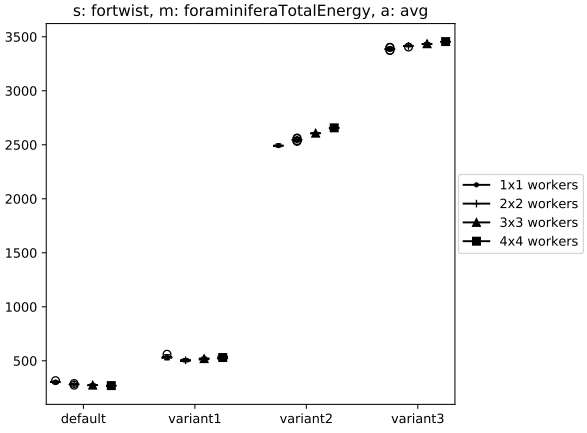

Figure 7. Simulation of foraminifera habitat: average total energy of foraminifera. Chart from [12].

On the other hand, the simulation of the predator-prey scenario yielded results that varied dramatically depending on the degree of distribution. The algorithm did not handle conflicts properly, as there was no possibility to resolve them in a way that would not violate the model. As a result, desynchronization led to the emergence of completely different behavior and inconsistent conclusions from the simulation.

In all three cases, the distribution and desynchronization led to substantial increases in the scalability of the simulations. In the two successful cases, the subtle differences in the measured values and observed results are definitely outweighed by the ability to assign many more computational resources. It is worth keeping in mind, however, that the simulation must be carefully analyzed and validated to ensure that the yielded results are useful.

5. Conclusion

In this paper, we have presented applications of the desynchronization concept to various algorithms. The concept aims to improve the scalability of the algorithms in HPC environments. It reduces the inter-node communication and limits synchronization-related hold-ups; however, it may affect the computational results. We have discussed two types of algorithms: simulations and optimization algorithms.

In the case of optimization, desynchronization might be applied to many algorithms with some kind of global knowledge; e.g., ant colony optimization, particle

swarm optimization, social cognitive optimization, or evolutionary multi-agent systems. The impact of desynchronization on the results is negligible and might be considered as another randomization factor. On the other hand, the speedup of the algorithm allows it to compute many more possible solutions of a problem, which leads to increased exploration and improved final results [18]. The speedup also enables faster optimization thanks to extensive parallelization.

Desynchronization might also be applied to simulations that involve multiple agents communicating with each other (e.g., the iterated prisoner's dilemma or gift-exchange game) or neighborhood-oriented global knowledge (like the board of Conway's Game of Life). The simulations require much more caution when subjected to desynchronization, since it might be the only randomization factor that affects the final results in an unacceptable manner. We have presented a simple experiment based on Conway's Game of Life, which has shown how desynchronization will significantly affect even such a simple simulation. When we introduced some desynchronization delays or lost updates, the results quickly became noticeably different.

On the other hand, the simulations of social processes are commonly randomized. In this case, desynchronization enables the system to simulate much bigger societies; however, one should be aware that the results might be affected by the specific algorithm construction. In the case of more-complex simulations, desynchronization might be successfully applied to models that heavily rely on randomness. There is no single strategy for handling such simulations; each desynchronized approach must be carefully analyzed and tested before being used as a proper model for further experiments.

To sum up, desynchronization is a concept that is applicable to multiple algorithms; however, it requires some caution (especially in the case of simulations where it might noticeably affect the results). Before drawing any further conclusions from the desynchronized version of a simulation, one should first estimate the influence of the introduced inaccuracies on the results. In optimization algorithms, we might assume that the introduced changes are negligible as long as we achieve comparable or better results when compared to the standard version. Hence, it is worth considering when a system requires high scalability in order to efficiently utilize HPC environments. One of the research areas related to optimization algorithms that has a strong potential of benefiting from introducing desynchronization is the domain of machine-learning algorithms. For example, it could be possible to desynchronize updating neuron weight matrices in neural networks, leading to improved efficiency and scalability.

Increasing scalability as a consequence of desynchronization actually does not require any explanation, being a simple consequence of applying Amdahl's or Gustafson's law [10] to predict the efficiency of concurrent and (in particular) distributed systems. However, one must ask the following question: to what extent does introducing desynchronization (thus allowing for some perturbations in the communication) impair the efficacy of the computing and simulation? We would like to try to answer this questions in one of our future works by constructing Markov chain models of possible perturbations and simulating several computing and simulation

cases similarly to well-known Markov-chain models of network packet loss (like the Simple Gilbert model [6]).

Acknowledgements

The research presented in this paper was supported by the Polish Ministry of Science and Technology funds assigned to AGH University of Science and Technology. The authors utilized the PLGrid infrastructure when conducting the experiments described in this work.

References

[1] Atashpendar A., Dorronsoro B., Danoy G., Bouvry P.: A scalable parallel cooperative coevolutionary PSO algorithm for multi-objective optimization, *Journal of Parallel and Distributed Computing*, vol. 112, pp. 111–125, 2018.

[2] Bujas J., Dworak D., Turek W., Byrski A.: High-performance computing framework with desynchronized information propagation for large-scale simulations. *Journal of Computational Science*, vol. 32, pp. 70–86, 2019.

[3] Byrski A., Dreżewski R., Siwik L., Kisiel-Dorohinicki M.: Evolutionary multiagent systems, *The Knowledge Engineering Review*, vol. 30(2), pp. 171–186, 2015.

[4] Conway J.: The game of life, *Scientific American*, vol. 223(4), p. 4, 1970.

[5] Dorigo M.: *Optimization, learning and natural algorithms*, PhD Thesis, Politecnico di Milano, 1992.

[6] Ellis M., Pezaros D.P., Kypraios T., Perkins C.: A two-level Markov model for packet loss in UDP/IP-based real-time video applications targeting residential users, *Computer Networks*, vol. 70, pp. 384–399, 2014, https://doi.org/10.1016/j.comnet.2014.05.013.

[7] Grakova E., Slaninová K., Martinovič J., Křenek J., Hanzelka J., Svatoň V.: Waste Collection Vehicle Routing Problem on HPC Infrastructure. In: *IFIP International Conference on Computer Information Systems and Industrial Management*, pp. 266–278, Springer, 2018.

[8] Ilie S., Bădică C.: Multi-agent approach to distributed ant colony optimization, *Science of Computer Programming*, vol. 78(6), pp. 762–774, 2013.

[9] Kennedy J., Eberhart R.: Particle Swarm Optimization. In: *Proceedings of ICNN'95 – International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[10] McCool M., Robinson A., Reinders M.: *Structured Parallel Programming: Patterns for Efficient Computation*, Elsevier, 2013.

[11] Nagano K., Collins T., Chen C.A., Nakano A.: Massively parallel inverse rendering using Multi-objective Particle Swarm Optimization, *Journal of Visualization*, vol. 20(2), pp. 195–204, 2017.

[12] 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.

[13] Skiba G., Starzec M., Byrski A., Rycerz K., Kisiel-Dorohinicki M., Turek W., Krzywicki D., Lenaerts T., Burguillo J.C.: Flexible asynchronous simulation of iterated prisoner's dilemma based on actor model, *Simulation Modelling Practice and Theory*, vol. 83, pp. 75–92, 2018.

[14] Ślaski M., Turek W., Gil A., Szafran B., Paciorek M., Byrski A.: Analysis of Distributed Systems Dynamics with Erlang Performance Lab, *Computer Science*, vol. 19, 2018.

[15] Starzec G., Starzec M., Byrski A., Kisiel-Dorohinicki M., Burguillo J.C., Lenaerts T.: Towards Large-Scale Optimization of Iterated Prisoner Dilemma Strategies. In: *Transactions on Computational Collective Intelligence XXXII*, pp. 167–183, Springer, 2019.

[16] Starzec M., Starzec G., Byrski A., Turek W.: Distributed ant colony optimization based on actor model, *Parallel Computing*, vol. 90, p. 102573.

[17] Starzec M., Starzec G., Byrski A., Turek W., Kisiel-Dorohinicki M.: Distributed ant system for difficult transport problems, *Journal of Intelligent & Fuzzy Systems*, vol. 37(6), pp. 7347–7356.

[18] Starzec M., Starzec G., Byrski A., Turek W., Piętak K.: Desynchronization in distributed Ant Colony Optimization in HPC environment, *Future Generation Computer Systems*, 2020.

[19] Xie X.F., Zhang W.J., Yang Z.L.: Social cognitive optimization for nonlinear programming problems. In: *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, pp. 779–783, IEEE, 2002.

Affiliations

Mateusz Starzec
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, mateusz.starzec@iisg.agh.edu.pl

Grażyna Starzec
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, grazyna.starzec@iisg.agh.edu.pl

Mateusz Paciorek
AGH University of Science and Technology, Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, al. Mickiewicza 30, 30-059 Krakow, Poland, mpaciorek@agh.edu.pl

Received: 26.04.2020
Revised: 28.06.2020
Accepted: 28.06.2020