Simulation Modelling Practice and Theory 128 (2023) 102812

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A methodology for selecting a performance-convenient ABMS development framework on HPC platforms

Andreu Moreno [a,b,*], Josep Jorba [c], Cristina Peralta [a], Eduardo César [a], Anna Sikora [a], Mauricio Hanzich [d]

[a] *Universitat Autònoma de Barcelona, Cerdanyola, Spain*
[b] *Escola Universitària Salesiana de Sarrià (EUSS), Barcelona, Spain*
[c] *Universitat Oberta de Catalunya, Barcelona, Spain*
[d] *Mitiga Solutions, Barcelona, Spain*

ARTICLE INFO

Keywords:
Agent-based modeling and simulation
Parallel applications
Performance
Benchmark
HPC

ABSTRACT

Agent-based modeling and simulation (ABMS) is an approach for simulating the actions and interactions of autonomous agents. Such interactions occur within a defined environment to assess their effects on a system as a whole. Depending on the complexity of the model and the number of simulated agents, an ABMS application may require a significant amount of computational resources. It makes them good candidates to be parallelized on HPC systems. However, most developers of ABMS simulators are experts in the specific simulation domain, but they lack the expertise to develop parallel applications. Consequently, several frameworks for generating HPC ABMS applications have been developed, and it may now be challenging for these non-expert users to choose which of these frameworks would provide the best performing simulator for a particular model. This paper presents a methodology that uses a benchmark to help non-expert users to select the most suitable framework to generate the best performing parallel implementation for a given ABMS model. Such a benchmark considers the common characteristics of parallel ABMS applications and includes parameters for influencing their relevant performance aspects. The methodology is based on defining a set of problem classes that represent the majority of known ABMS models and systematically conducting a series of experiments to determine which framework offers the best performance for each class. Then, users only need to identify the class that closely aligns with their model to make an informed decision regarding the appropriate development framework. The methodology is used to assess well-known ABMS parallel development frameworks (FLAME, RepastHPC, and DMASON) on real HPC platforms. The obtained results are validated using a real application for infection and contact tracing modeling.

1. Introduction

Agent-based modeling and simulation (ABMS) is an approach for simulating the actions and interactions of autonomous agents within a defined environment to assess their effects on a system as a whole [1]. The agents' behavior is realized through algorithms, ranging from simple heuristics to advanced learning and adaptive strategies. An essential aspect of this behavior involves interaction

* Corresponding author at: Escola Universitària Salesiana de Sarrià (EUSS), Barcelona, Spain.
E-mail addresses: andreu.moreno@uab.cat, amoreno@euss.cat (A. Moreno), jjorbae@uoc.edu (J. Jorba), cristina.peralta@uab.cat (C. Peralta), eduardo.cesar@uab.cat (E. César), anna.sikora@uab.cat (A. Sikora), mauricio.hanzich@mitigasolutions.com (M. Hanzich).

https://doi.org/10.1016/j.simpat.2023.102812
Received 6 March 2023; Received in revised form 9 June 2023; Accepted 25 July 2023
Available online 29 July 2023

with other agents, as it enables agents to influence each other's behavior, thereby generating complex interactions within the system. ABMS has become a popular modeling approach in fields such as social media [2,3], traffic flow management [4,5], distribution systems [6,7], economy [8], biology [9] or medicine [10], and it has novel application scenarios in IoT systems [11,12].

The use of ABMS started with models composed of a few agents whose behavior was ruled by simple heuristics or analytical expressions. These models could be executed in a single workstation. However, in current models simulating complex systems, the number of agents can be very high, the agent behavior can be implemented by time-consuming algorithms, and the communication among them may imply a high volume of data. This can lead to a high computational burden, requiring increasingly more computing resources. Therefore, these simulations can take advantage of parallel techniques and High Performance Computing (HPC) to efficiently handle the computational demands.

In the last 20 years, several frameworks to implement ABMS on HPC systems have been developed [13]. According to several studies, such as [14] or [15], and their use on real cases, such as [16,17], among the most relevant frameworks, we have FLAME [18], FLAME GPU [19], RepastHPC [20], EcoLab [21], DMASON [22], Pandora [23] or Care-HPS [24]. The shared objective of all these frameworks is to hide the difficulties of developing parallel applications from the typical ABMS developer, which is usually an expert on the specific simulation domain but lacks the expertise to develop parallel applications. Nevertheless, these frameworks present differences in the way agents and contexts are specified and implemented (e.g., C, C++, Java, XML, Tcl), in how communications are managed (e.g., explicit messages, agent replication), in the presence or not of a load balancing mechanism, among others. Due to these differences, the performance of the resulting simulator can vary significantly across different hardware platforms and for different problems being addressed.

Therefore, choosing a framework for implementing a parallel ABMS is not straightforward for these non-expert users. To simplify this task, a benchmark for comparing the performance of parallel ABMS development frameworks was defined in [25]. This work presented the design of a benchmark that could be parameterized to take into consideration the computation load and distribution, the communication load and pattern, and the problem size. It also introduced an initial benchmark implementation using RepastHPC, FLAME, FLAME GPU, and EcoLab. However, this work did not introduce a systematical methodology for using this benchmark to choose the appropriate framework on a given hardware platform.

In this paper, we build upon our previous contribution to define such a methodology and introduce extensions and improvements to the benchmark. Specifically, the contributions of this paper can be summarized as follows:

- Definition of a systematical methodology for selecting the performance-convenient HPC ABMS development framework for a given problem and hardware platform. This methodology is based on defining a set of representative problem classes and the synthetic implementation of these classes using the developed benchmark.
- Presenting the implementation of the benchmark for DMASON.
- Using the methodology for assessing some of the most relevant development frameworks (FLAME, RepastHPC, and DMASON) on real HPC platforms.
- Applying the methodology results to a real case study.

The remainder of this work is organized as follows. Section 2 summarizes previous work and presents the benchmark's improvements and extensions. Next, Section 3 introduces the defined methodology. Section 4 introduces a comprehensive assessment of RepastHPC, FLAME, and DMASON in a real HPC system using the proposed methodology. Then, Section 5 applies the obtained results to a real case study. Section 6 presents some related works. Section 7 critically discusses the main results of this work and, finally, conclusions and future work are stated in Section 8.

2. Benchmark evolution

We introduced the design and implementation of a benchmark for assessing the performance of parallel ABMS models on HPC platforms in [25]. This study focused on determining the general features of real ABMS models that significantly impact the performance of their simulation. Then, a benchmark was designed as a parameterizable synthetic model, which could be tailored to match the performance characteristics of real models. Finally, it was implemented using a set of well-known frameworks for developing parallel ABMS.

Unsurprisingly, the two main features influencing the performance of a parallel ABMS are (1) the frequency and volume of interactions between agents and (2) the amount of computation performed by each agent.

On the one hand, every framework offers some kind of abstraction to express interactions between agents. For example, FLAME offers a data structure reachable for all agents where every message is stored. At the same time, RepastHPC manages agents' lists representing neighbors, and communications are done by calling methods of these proxy agents. These abstractions are implemented using a mechanism that allows to communicate and synchronize threads and/or processes on the target hardware system; in the case of FLAME and RepastHPC, it is Message Passing Interface (MPI) [26], and in the case of DMASON, it is ActiveMQ. Logically, this communication and synchronization mechanism introduces an overhead affecting simulation performance.

Usually, agents interact with their neighbors frequently and interchange small amounts of information. However, sometimes agent interactions can be between any pair of agents, not necessarily neighbors, or can imply interchanging larger amounts of information. Consequently, to assess the communication and synchronization overhead, the benchmark must be able to reproduce the usual interaction patterns among agents and allow for configuring these interactions (pattern, frequency, size).

On the other hand, the computation done by each agent is coded in functions or methods in the framework. The overall ABMS performance is influenced by the global workload, how this computation is balanced among processes, and its ratio against

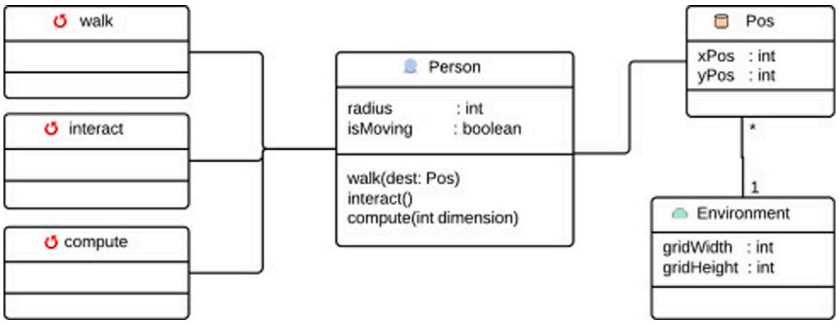


Fig. 1. AML representation of the base agent model.

communication/synchronization. Usually, the amount of computation performed by an agent between consecutive interactions is relatively small, but in other cases, the amount of computation can be significant. Therefore, the benchmark must be able to represent different workloads, agents distribution, and communication/computation ratios.

The benchmark contains the following configurable parameters to represent all of these potential behaviors: *communication volume* and *communication pattern* among agents, *computation load* of agents, the *initial number* of agents and its *distribution*, *death and birth rates*, *death and birth centers*, *size of the environment*, and the *number of computing elements*.

The final design of the agent is based on the adaptation done in [14] of the *prisoner's dilemma* implementation used in a RepastHPC demo example. Fig. 1 shows the agent model language (AML) [27] representation of this base agent model.

The agent behavior is divided into the three following states, executed in each simulation step:

1. The *walk state* allows agents to move randomly on the environment to one of its 8 Moore neighborhood cells in the grid. This state affects the agent *distribution*.
2. The *interact state* allows agents to interact with num_agents agents in their perception field (determined by $distance$), which determines the *communication pattern* among agents. The message comprises the agent id and an integer value plus a num_bytes number of arbitrary bytes, which determine the *communication volume*.
3. The *compute state* simulates the *computation load* generated by the agent's inner algorithms. This state computes a one-dimensional Fast Fourier Transform (FFT) [28] on a table of $table\_sz$ elements.

An initial implementation of the benchmark for RepastHPC, FLAME, FLAME GPU, and EcoLab was presented in [25], along with experimental results in a small HPC system. Since then, the following improvements and extensions have been added to the benchmark[1] :

- The benchmark incorporates the use of GNU Autotools to simplify the installation and configuration process. Additionally, it includes integration with the TAU [29] instrumentation tool.
- A new tool to visualize the spatial agent distribution in any simulation step.
- The implementation of the benchmark for the DMASON framework, which is described in the next section.
- Experimentation conducted in real HPC systems.

We have also considered implementing the benchmark for Pandora [23] and Care-HPS [24], but it was not possible due to the limitations in porting them to the production environments. Another good option for implementing the benchmark could be Distributed MASON [30], which attempts to develop a better MASON version for HPC systems using MPI. Distributed MASON can run over local clusters or cloud computing architectures (such as Amazon Web Services). However, a functional version of Distributed MASON was not available when the experimental work was being conducted for this research.

2.1. Benchmark implementation in DMASON

DMASON [22] is a distributed ABMS development framework written in Java and derived from the sequential ABMS development framework called MASON [31]. It is based on the Master-Worker paradigm and uses a space partitioning approach.

DMASON architecture is composed of four components:

- A Master process that partitions the simulated space (*field*) into *regions* and assigns each region, along with the agents it contains, to a slot in a Worker process.

[1] The implementation of benchmark for different platforms is available at https://github.com/HPCA4SE-UAB.

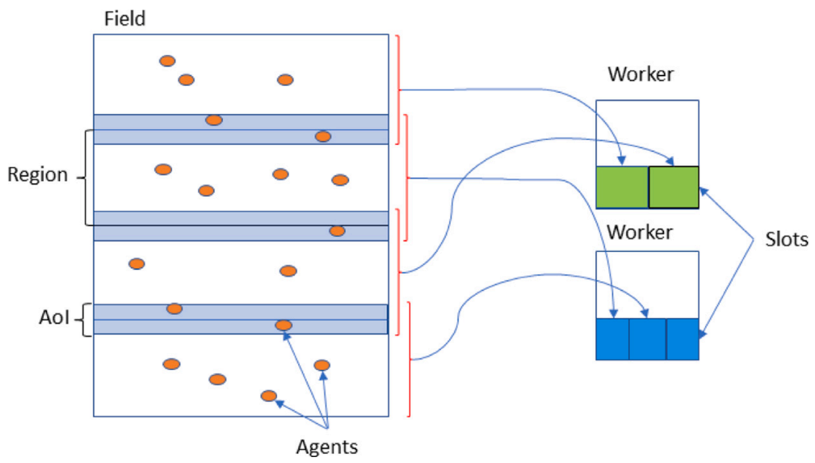


Fig. 2. DMASON partitioning of the simulated field into regions and assignment of these regions to available workers.

- A set of Worker processes, which are responsible for simulating the agents in the assigned regions, handling the migration of agents moving among regions, and managing the synchronization between neighboring regions (Area of Interest - *AoI*). Each Worker can be configured with the maximum number of regions it can simulate (number of *slots*) according to its computational capabilities. Fig. 2 illustrates how the simulated *field* is partitioned into *regions* sharing an *AoI*, and how these regions are assigned to Worker *slots*.
- An ActiveMQ server that handles communications among Workers.
- A Web User Interface called System Management, which manages the environment and simulation configuration, as well as visualizes simulation output.

Each DMASON Worker iteratively simulates the evolution of the agents in its assigned regions. Each simulation step consists of two phases:

1. Communication/Synchronization phase. Workers send their neighbors information regarding agents migrating to their regions and agents present on their regions' AoI.
2. Simulation Phase. Each Worker computes the agent step() function for every agent present in their regions.

The default communication protocol in DMASON is JMS, provided by an Apache ActiveMQ[2] server running in the same node as the DMASON Master process. Communication follows a Publish-Subscribe protocol, where agents propagate their state information using multicast channels assigned to each region that will be shared. Workers subscribe to the communication channels of the regions that overlap with their AoI. For every simulation step, agents located inside the boundary region determined by the AoI are sent by the Worker responsible for the region to their corresponding topic on the ActiveMQ server, where the neighbor Workers download them before the next simulation step.

According to the described architecture, the benchmark [25] has been adapted to DMASON, implementing the following classes:

- **DPrisDilemma**. Model code for the simulation that represents the distributed environment to be simulated (extends D-MASON DistributedState class).
- **DRemotePrisoner**. An abstract class that implements DMASON's *RemotePositionedAgent* class and Java's *Serializable* class. It contains the agent's unique identifier as well as its current position in the field.
- **DPrisoner**. It extends *DRemotePrisoner* class with the agent's logic.

The agent's logic has been implemented in the following methods:

- **Walk**, which implements the *walk state*. The agent changes its position, moving to a new one in its Moore neighborhood.
- **Play**, which implements the *interact state*. Agents play with num_agents neighbors that are within a maximum radius determined by $distance$, sending messages of $communication\ volume$ bytes.
- **Compute**, which implements the *compute state*. It simulates the agents' workload by performing an FFT on a table of $table\_sz$ elements.

[2] ActiveMQ link https://activemq.apache.org/.

- **Reproduce**, which allows changing the agents' distribution and total number. An agent may generate a new agent with a certain non-uniform probability (parameter *birth rate*). This probability decreases linearly according to the agent's distance from a given position in the field (parameter *birth center*).
- **Die**, which allows changing the agents' distribution and total number. An agent may die with a certain non-uniform probability (parameter *death rate*). This probability decreases linearly according to the agent's distance from a given position in the field (parameter *death center*).

Additionally, we made the following modifications to the original DMASON 3.2 code to solve some issues detected during the benchmark development and testing in HPC systems[3]:

- Changes in the *run* method of *CellExecutor* class to use the override *finish* method to check the simulation results.
- Changes in the detection of network interfaces. In the original code, when an IP address is provided, DMASON looks first for an ethernet interface. However, in an HPC platform, the main interface (for very low latency communications) is usually Infiniband. Consequently, the code has been adapted to look for the Infiniband interface first.

3. Methodology for assessing ABMS development frameworks on an HPC platform

The developed benchmark can be used to decide which framework would lead to the best performance for a given ABMS model on a given HPC platform without actually implementing the model. The user only needs to set the benchmark parameters accordingly to the model characteristics. These parameters are listed in Section 2 (*communication volume*, *communication pattern*, *computation load*, *the initial number of agents*, etc.). Then, the user can run the benchmark as many times as needed using the different available frameworks (e.g., FLAME, RepastHPC, and DMASON). Finally, the user can compare performance results and choose the framework leading to the best performance for the given model and HPC platform.

However, this benchmark can also serve as a valuable tool for conducting a comprehensive performance analysis of a set of development frameworks on a specific HPC platform. The basic underlying idea is that it is possible to predefine a finite set of synthetic classes that are representative of most ABMS and then systematically assess the performance of these classes to know which is the best framework for each case. Consequently, users will not even need to characterize and test the benchmark; they will only need to find which is the class that best matches their ABMS model and use the development framework that is known to provide the best performance.

The main objective of this paper is to define a systematical methodology for achieving this comprehensive performance characterization. Therefore, we will first discuss how to define the representative classes and next describe how to use them to assess the performance of a framework on a given HPC platform.

3.1. Defining representative ABMS classes

To define the set of representative ABMS classes, we have explored different areas where ABMS has been applied. Review paper [32] shows with examples that ABMS has been used, among others, in the following areas: agriculture, air traffic control, anthropology, biomedicine, crime analysis, ecology, energy analysis, epidemiology, evacuation, market analysis, organizational decision making, and social networks.

It can be observed that agents' behavior is usually modeled with some heuristics or analytical expressions, which leads to a low computation load. However, complex systems, such as the simulations associated with system biology (e.g., [33]), usually lead to high computation loads. In addition, it can also be seen that usually, agents interact by communicating a small amount of data only with surrounding agents. Nevertheless, in some ABMS applications, such as the simulation of the European economy [16], agents communicate hundreds of bytes of data with other agents independently of the distance between them.

From this exploration, we have derived 9 ABMS generic classes with different computation and communication loads, which represent most ABMS models. Table 1 shows the nine considered classes. As described in Section 2, the computation load is simulated by computing the FFT function using inputs of different sizes (8, 16 Ki, and 128 Ki elements for low, medium, and high loads, respectively). Communication load is simulated by interchanging messages of different sizes among neighboring agents (4 B, 256 B, and 2 KiB for low, medium, and high loads, respectively).

Regarding the computation load, calculating an FFT on a table of 8 elements may correspond to solving a few algebraic expressions; calculating an FFT on a table of 16 Ki elements can be compared to solving a small system of equations; and doing so on a table of 128 Ki elements can be compared to the evaluation of a machine learning model.

Regarding the communication load, a message of 4B could represent one interaction for communicating a simple state of the agent (e.g., infected-not infected or the agent's position in the space); a message of 256B could represent one interaction for communicating information involving several elements, such as economic transactions; and a message of 2 KiB may represent one interaction for communicating a whole set of knowledge, such as the set of beliefs of an individual, a genealogy, or a set of assets.

[3] See GitHub repository: https://github.com/HPCA4SE-UAB/ABMS-Benchmark-DMASON.

Table 1
Derived 9 ABMS classes with different computation and communication loads.

	High	2 KiB	H-L	H-M	H-H
Comm.	Medium	256B	M-L	M-M	M-H
	Low	4B	L-L	L-M	L-H
			8	16 Ki	128 Ki
			Low	Medium	High
			Computation		

3.2. Assessing framework performance

In order to assess a given ABMS development framework performance, we analyze the performance characteristics of the parallel simulator generated by the framework for each of the nine classes defined previously. We propose to:

1. Perform a strong and weak scalability analysis of the execution of the simulator,
2. Analyze the reliability of the simulator, stressing the simulator using a wide range of agents for a fixed number of cores,
3. Analyze the performance of the simulator under heavy load imbalance conditions.

Although their specific values and ranks will depend on the HPC platform being used, the following parameters must be set depending on the type of analysis:

1. **Strong scalability**: *total number of agents*, *minimum and maximum number of cores*,
2. **Weak scalability**: *minimum and maximum number of cores*, the *minimum number of agents* that will be scaled in the same proportion as the number of cores,
3. **Reliability analysis**: *minimum and maximum number of agents* and *fixed number of cores*,
4. **Heavy load imbalance**: the factors determining the **imbalance degree** (agents' *birth and death rates* and *birth and death centers*).

In addition, the specific parameters associated with each framework must be also considered. In this study, we have used RepastHPC, which does not include any specific parameter; FLAME, which allows for configuring how the agents are distributed among the simulation processes (geometric or Round-Robin partitioning); and DMASON, which allows the user to set the number of slots for each Worker.

Each resulting configuration is executed a certain number of times to guarantee statistical validity (up to 5 times in our case, as the standard deviation is small enough), and the average execution time is used to assess the framework.

At the end of this process, summarized in Fig. 3, the framework that offered the best performance and maximum reliability for each of the nine classes of the evaluated HPC platform will be known. Now, a user that wants to implement an ABMS and knows the characteristics of the model can use this information to decide the most adequate framework to implement it on this platform. This task is challenging because it requires a good knowledge of the framework, but the user is now more confident in the expected performance of the resulting simulator.

4. Using the methodology on a real HPC system

The initial idea of this work was compare the performance of the simulators generated by the frameworks FLAME, RepastHPC, and DMASON on the same HPC platform. This was the first step in evaluating these frameworks for implementing an ABMS simulating epidemic infection spread and contact tracing systems. This evaluation has been done in the context of the Epidemic Early Warning System (Epi-EWS) project, which was granted 5 million computation hours by the PRACE call *Support to mitigate the impact of the COVID-19 pandemic*.

The simulations were to be executed in the Beskow computer [34] of the Swedish PDC Center for High Performance Computing. It is a CRAY XC40 system based on Intel Haswell and Broadwell processors and Cray Aries interconnect technology (Dragonfly topology), with 2060 compute nodes, 67,456 cores in total, and a theoretical peak performance of 2.43 petaflops. We have used nodes with 2 Intel CPUs Xeon E5-2698v3 Haswell 2.3 GHz CPUs (16 cores per CPU), and 64 GB of main memory (RAM).

Beskow was adequate to execute simulators generated with FLAME and RepastHPC. However, due to some of the framework's specific characteristics, it was not possible to execute the simulators generated with DMASON on this particular platform.

As mentioned in Section 2.1, DMASON uses a web interface (System Management), which needs a tunneled ssh connection to the DMASON Master process during the simulation. This kind of connection is not allowed in Beskow.

Nevertheless, the PDC Center offered the possibility of executing DMASON experimentation in the Tegner supercomputing platform. Tegner is used primarily as a preprocessing and/or postprocessing system for Beskow. However, it can also be used as a general cluster for executing small to medium size parallel programs. Tegner offers the necessary external connectivity capabilities and a higher main memory capacity in each node, which is important for scaling DMASON simulations.

Tegner is a heterogeneous system with 67 nodes based on Intel Haswell and Ivy Bridge CPUs and Nvidia GPUs, up to 2 TB of RAM memory/node, and an EDR Infiniband interconnection. Its peak performance is 66 teraflops (only CPUs). We have used Tegner nodes (thin nodes) with 2 CPUs Intel E5-2690v3 Haswell (12 cores per CPU), and 512 GiB of main memory (RAM).

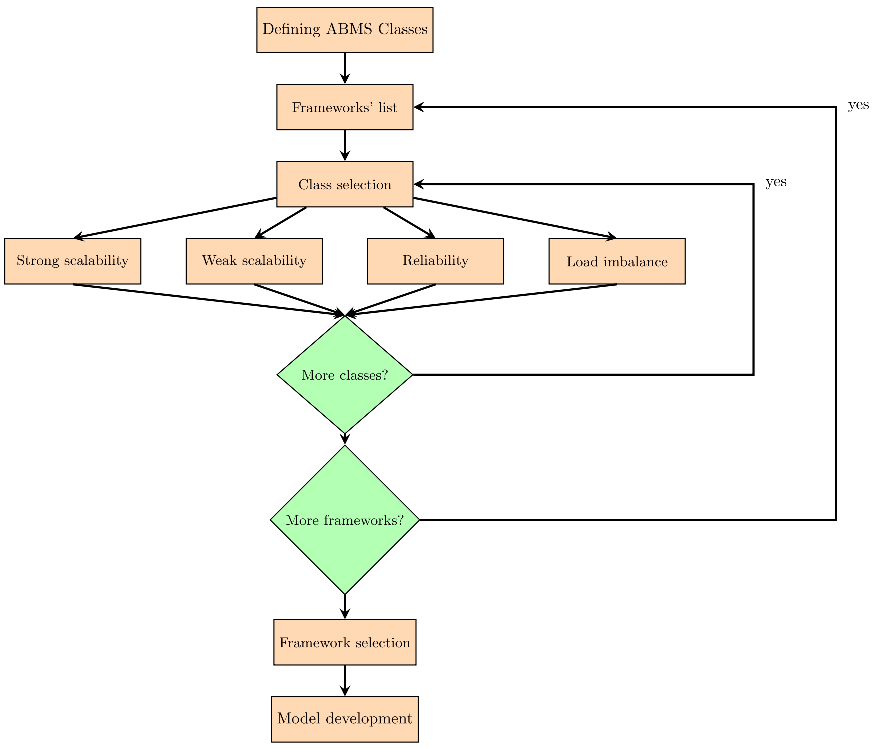


Fig. 3. Agent-based model simulation methodology.

Table 2
Summary of the chosen parameters' values for the performance analysis of FLAME and RepastHPC on Beskow.

Analysis	Cores	Agents	Birth rate	Death rate	Classes
Strong scaling	**128 .. 4 Ki**	16 Ki	0.2	0.2	L-L .. H-H
Weak scaling	**128 .. 4 Ki**	**8 Ki .. 256 Ki**	0.2	0.2	L-L .. H-H
Reliability	512	**1 Ki .. 256 Ki**	0.2	0.2	L-L .. H-H
Birth and death	256	16K	**0.0, 0.2, 0.6**	**0.0, 0.2, 0.6**	M-M

The proposed methodology will evaluate a framework for a given problem and a specific hardware platform. Consequently, since the experimentation for FLAME and RepastHPC has been performed on a different platform than DMASON, we present the corresponding results in two subsections.

4.1. Assessing FLAME and RepastHPC on Beskow

When applying the methodology introduced in Section 3, the first step consists of setting the parameters associated with each type of analysis for the target HPC platform. For Beskow, the following values, summarized in Table 2, have been set:

1. **Strong scalability**: *total number of agents* = 16 Ki, *minimum and maximum number of cores* = 128 and 4 Ki, respectively,
2. **Weak scalability**: *minimum and maximum number of cores* = 128 and 4 Ki, respectively, *minimum number of agents* = 8 Ki, which scales up to 256 Ki in the same proportion as the number of cores (32X),
3. **Reliability analysis**: *minimum and maximum number of agents* 1 Ki and 256 Ki, respectively, and *fixed number of cores* = 512,
4. **Heavy load imbalance**: for the agents' *birth and death rates*, 0.0, 0.2, and 0.6. For the *birth and death centers*, in a simulation area of 1024×1024, the birth center has been set at 750, 750 and the death center at 250, 250.

Strong scalability analysis

Strong scaling analysis implies the simulation of the 9 ABMS classes summarized in Table 1 with 16K agents on 128, 256, 512, 1024, 2048, and 4096 cores. Each simulation consists of 40 simulation steps and has been executed 5 times using FLAME (with geometric and Round-Robin partitioning) and RepastHPC. Results have shown that the average of 5 simulations is enough because the standard deviation is below 5% of the average execution time value.

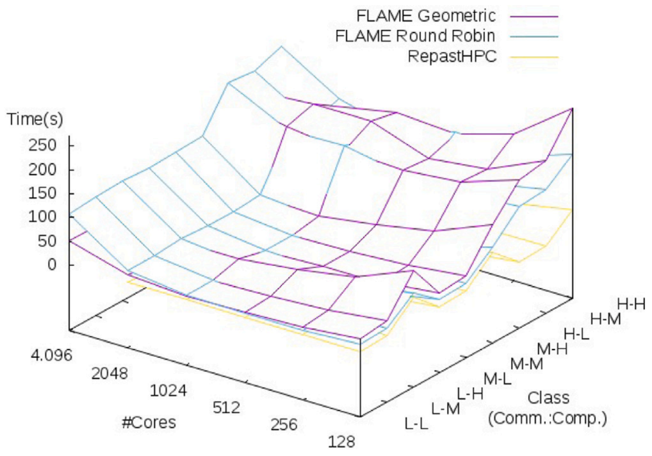
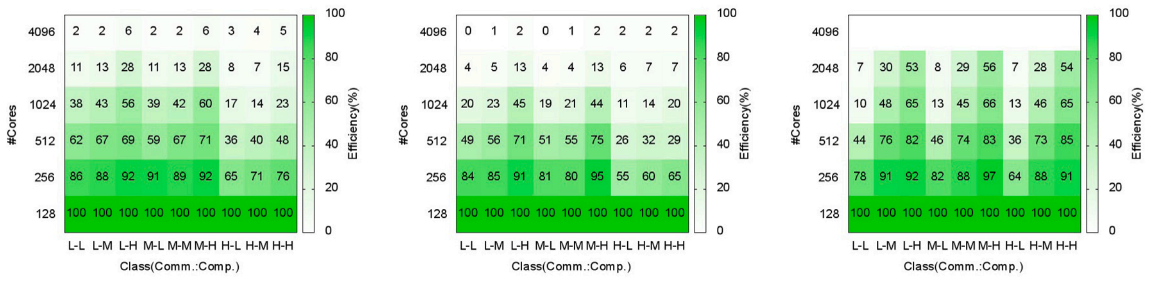


Fig. 4. Strong scaling analysis results: FLAME with geometric partitioning, FLAME with Round-Robin partitioning, and RepastHPC.



(a) FLAME geometric (b) FLAME Round-Robin (c) RepastHPC

Fig. 5. Strong scalability analysis: Efficiency index for each assessed simulator.

Fig. 4 shows the average execution time of each of the 54 simulation cases for each of the 3 simulators (generated by FLAME Round-Robin, FLAME geometric, and RepastHPC). It can be seen that the simulator generated by RepastHPC outperforms the one generated by FLAME in all the cases, especially for the ABMS classes with the highest communication volume (H-L, H-M, and H-H). Given that the performance difference is exacerbated when the communication volume and the number of cores increase, it can be concluded that the fundamental reason for this difference is the centralized communication mechanism of FLAME.

It can also be observed that, for smaller numbers of cores, Round-Robin partitioning outperforms geometric one, while for higher numbers of cores, it is the other way around. Geometric partitioning may lead to load imbalances because the assignation of agents to cores depends on their position in the space. In contrast, Round-Robin partitioning always leads to balanced loads because it uniformly distributes the agents. When the number of cores is small (e.g., 128), the number of agents assigned to each core is relatively high (e.g., 128 agents/core in the case of Round-Robin partitioning); thus load imbalances introduced by the geometric partitioning decrease the performance of the simulation. However, when the number of cores is high (e.g., 4096) the number of agents assigned is very small (e.g., 4 agents/core in the case of Round-Robin partitioning, while for geometric partitioning, many cores may have 0 agents assigned). Consequently, in this case, the overhead introduced by the centralized communication mechanism of FLAME is having a stronger effect on the simulation using Round-Robin because all the cores are participating in the communication.

To individually assess the performance of each of the 3 simulators, we have calculated the efficiency of each simulation case in relation to its execution with 128 cores. We have used Eq. (1) to make this calculation, which assumes that the execution using 128 cores has an efficiency of 100%.

Fig. 5 shows the value of this index in the form of a heatmap for each of the simulators. It can be seen that the simulator generated by RepastHPC shows the expected behavior, i.e., the efficiency decreases when the number of cores is increased, and this decrement is smaller when the computation load is higher. In this case, we can also see that for 4 Ki cores, this simulator crashed most times; this issue will be discussed later in the reliability analysis. For the simulators generated with FLAME, this expected behavior can be seen for up to 1024 cores and the ABMS classes with low and medium communication volumes. For higher numbers of cores and the highest communication volume, the efficiency sharply drops. Once again, this is caused by the overhead introduced by FLAME's centralized communication mechanism.

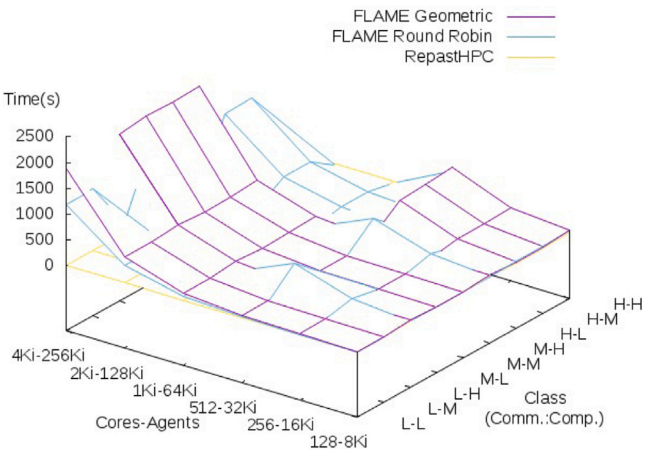
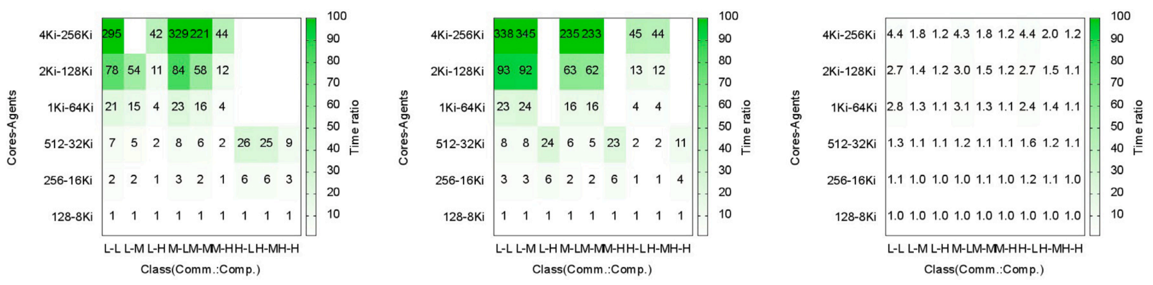


Fig. 6. Weak scaling analysis results: FLAME with geometric partitioning, FLAME with Round-Robin partitioning, and RepastHPC.



(a) FLAME geometric (b) FLAME Round-Robin (c) RepastHPC

Fig. 7. Weak scalability analysis: Time ratio in weak scaling scenario.

$$Efficiency = \frac{128 \times T_{128}}{n \times T_n} \tag{1}$$

Weak scalability analysis

Weak scaling analysis implies the simulation of the 9 ABMS classes summarized in Table 1 increasing the number of agents in the same proportion as the number of cores. It led to the following configuration pairs (number of cores, number of agents): (128, 8 Ki), (256, 16 Ki), (512, 32 Ki), (1 Ki, 64 Ki), (2 Ki, 128 Ki) and (4 Ki, 256 Ki). Again, each simulation consists of 40 simulation steps and has been executed 5 times using FLAME (with geometric and Round-Robin partitioning) and RepastHPC.

Increasing the size of the problem in the same proportion as the number of resources leads to a constant workload per core. Consequently, in the ideal case, the execution time of the parallel application should be the same for all cases. Fig. 6 shows the average execution time of each of the 54 simulation cases for each of the 3 simulators (generated by FLAME Round-Robin, FLAME geometric, and RepastHPC). It can be seen that in the case of FLAME, the results are far from the ideal, especially for executions with a higher number of agents, while for RepastHPC, the results seem closer to the ideal.

In order to analyze each simulator individually, Fig. 7 shows the evolution of the ratio between the execution time of the 54 simulation cases in relation to the (128, 8 Ki) configuration. In the ideal case, this ratio should always be 1.0. It can be seen that RepastHPC scales well for most of the cases because this ratio is always smaller than 2, except for the biggest simulations with low computation (ratio up to 4.4). On the contrary, FLAME generated simulators are not scalable. It is so because, for the executions using more than 1 Ki cores, the ratio can reach up to 345 (Round-Robin partitioning), and several configurations crash because there is insufficient memory to satisfy the application requirements.

Reliability analysis

For analyzing the reliability of the simulators generated by FLAME and RepastHPC, we have designed an experiment consisting of simulating the 9 ABMS classes summarized in Table 1 using a fixed number of cores (512) and varying the number of agents from 1 Ki to 256 Ki (doubling the number of agents in each configuration). This way, we are testing cases from a very low to a

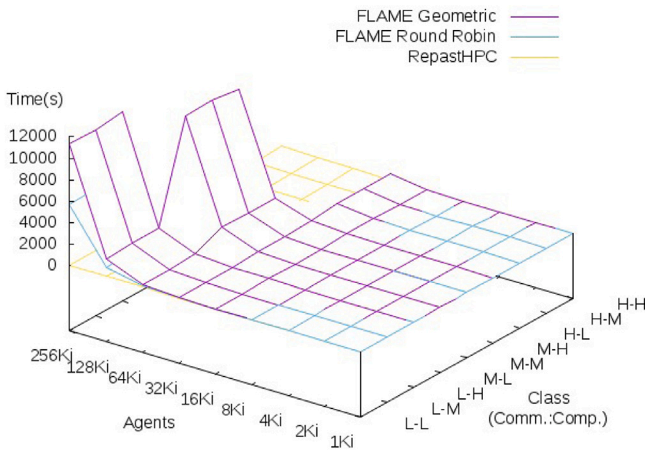
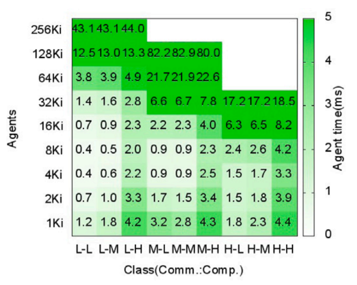
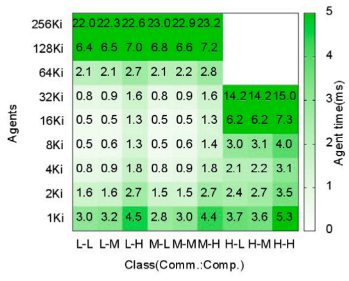


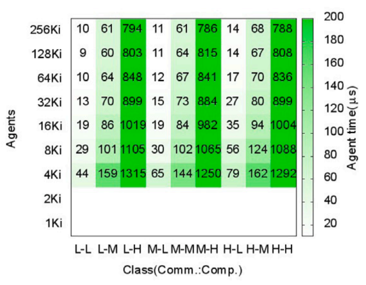
Fig. 8. Reliability analysis: FLAME geometric, FLAME Round-Robin, and RepastHPC.



(a) FLAME geometric(ms)



(b) FLAME Round-Robin (ms)



(c) RepastHPC(μs)

Fig. 9. Reliability analysis: Average agent execution time.

high computation load per core. Each simulation consists of 40 simulation steps and has been executed 5 times using FLAME (with geometric and Round-Robin partitioning) and RepastHPC.

Fig. 8 shows the average execution time of each of the 81 simulation cases for each of the 3 simulators (generated by FLAME Round-Robin, FLAME geometric, and RepastHPC). RepastHPC fails for the two smallest numbers of agents (1 Ki and 2 Ki) because there are no agents in some of the cores. It happens because the generated simulator executes the simulation loop regardless of the number of agents; thus developers must control the presence of agents explicitly. This behavior has already been observed in the strong scalability analysis (Fig. 5) when executing 16 Ki agents using 4 Ki cores. FLAME fails for a bigger number of agents and the highest communication volume. As discussed in the weak scalability analysis (Fig. 7), in these cases, the simulator is depleting the main memory available, and, consequently, the program crashes because there are no memory checks.

To analyze each simulator independently, we have calculated, for each configuration, the average simulation time per agent using Eq. (2). Fig. 9 shows the value of this index in the form of a heatmap for each simulator. In this figure, it is evident which configurations have failed as they lack corresponding results. In addition, given that FLAME does not scale well when growing the problem size beyond 16 Ki agents, the average time per agent increases sharply because the available resources become increasingly saturated. On the contrary, RepastHPC, which has significantly better scalability, shows the expected behavior. The average agent time decreases when growing the problem size because the relative weight of the inherently serial component of the simulation also decreases.

$$Average\_agent\_time = \frac{T_{exec}}{number\_of\_agents} \tag{2}$$

Heavy load imbalance

Finally, for analyzing the effects of heavy load imbalances on the simulation performance, we have executed the medium computation load/medium communication volume (M-M) ABMS class using 256 cores, 16 Ki agents, and three different rates of agents' birth and death. The 0.0 birth and death rate indicates no imbalance, the 0.2 rate indicates a low imbalance, and the 0.6 rate indicates a significant load imbalance. Fig. 10 shows the results for each of the tested cases. Even though it does not include an

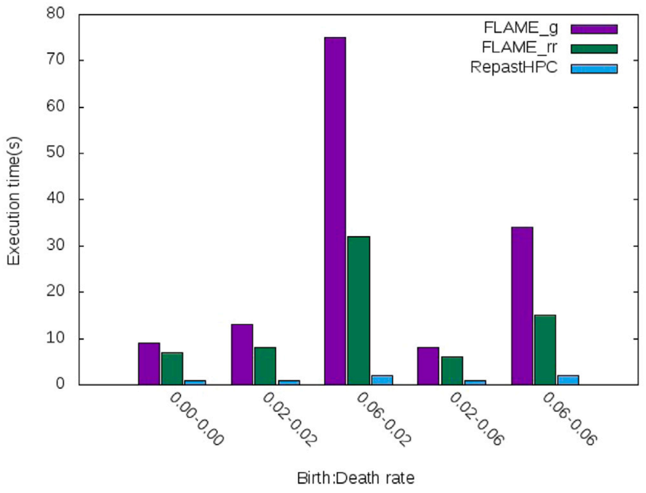


Fig. 10. Load imbalance analysis: FLAME geometric, FLAME Round-Robin, and RepastHPC.

automatic load balancing mechanism, RepastHPC is not strongly affected by the introduced imbalances. At the same time, FLAME presents wide variations in the execution time, especially in the case of using geometric partitioning. It is because, for geometric partitioning, the agents with the highest probability of reproducing are concentrated in a few nodes. Consequently, the demand for resources in those nodes is sharply increasing because, as we have seen before, FLAME does not scale well.

4.2. *Assessing DMASON on tegner*

When applying the methodology introduced in Section 3, the first step consists of setting the parameters associated with each type of analysis for the target HPC platform. For Tegner, the following values have been set:

1. **Strong scalability**: analyzing for a *total number of agents* of 1 Ki, 2 Ki, 4 Ki, and 8 Ki, using *minimum and maximum number of cores* of 24 and 192 (1 to 8 worker nodes), respectively,
2. **Weak scalability**: *minimum and maximum number of cores* = 24 and 192, respectively, *minimum number of agents* = 1 Ki, which scales up to 8 Ki,
3. **Reliability analysis**: *minimum and maximum number of agents* 1 Ki and 1024 Ki, respectively, and analyzing for *a fixed number of cores* of 24, 48, 96 and 192,
4. **Heavy computation load**: for a fixed high number of agents of 512 Ki, change the computation load of each agent, calculating the FFT on a *table size* of 8B, 16 Ki, 32 Ki, 64 Ki, and 128 Ki elements.

In the case of DMASON, there are several framework-specific issues to consider in the performance assessment. First, DMASON requires setting the number of slots for each Worker. This parameter significantly influences the performance of the simulator, and hence it should be analyzed and tuned for the given HPC system before applying the methodology. In addition, DMASON has specific characteristics that are better suited for distributed systems but make the configuration of the simulator execution in HPC platforms more difficult:

- The Web User Interface (System Management) is used to configure simulation executions interactively. Consequently, resources must be first obtained interactively (e.g., via SLURM interactive jobs) and then explicitly linked to the Web User Interface.
- The Web User Interface forces to manually set the parametric configuration of each experiment, i.e., this configuration cannot be automatically done (e.g., via a SLURM script).

These characteristics explain why we limit each computational experiment to a small number of nodes (from 1 to 8, i.e., from 24 to 192 cores). Moreover, they also restricted the possibility of executing all the experiments for the 9 ABMS classes defined in Table 1. Consequently, we have used the M-M class (256B message size and 16 Ki FFT computation size) in all analyses. As this decision eliminates the ABMS class dimension shown in the study made on Beskow, we have added an extra experiment, previously defined as *Heavy computation load*, to visualize this dimension in a particular case by fixing the number of agents and using different computation loads per agent (changing FFT Size).

Summarizing, for the performance analysis of DMASON on Tegner, we will follow these steps:

- Describe the deployment of a distributed ABMS Java-based system (DMASON) in an HPC supercomputer (Tegner).
- Determine the best number of slots per Worker for conducting all the methodology experiments.
- Apply the proposed methodology (analysis of strong and weak scalability and reliability) using only the M-M ABMS class.

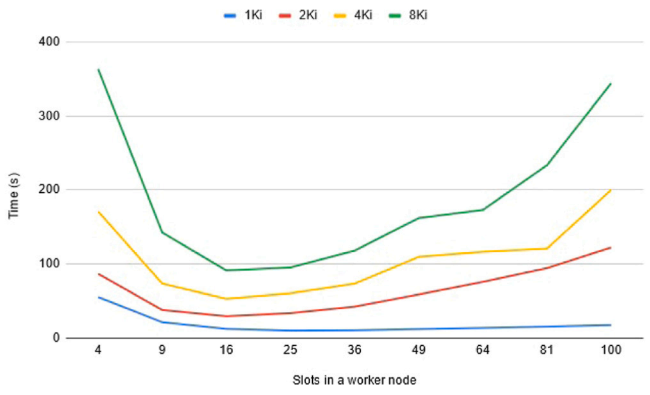


Fig. 11. Impact of the number of slots available in a Worker node in the execution time of the simulation for different numbers of agents.

- Analyze the effect of changing the agents' computation load in the simulation performance in order to visualize the class dimension (M-L, M-M, and M-H classes) for a fixed number of agents.

Deploying DMASON

For deploying a simulator generated with DMASON in an HPC environment, it is necessary to allocate compute nodes for the Master and Worker processes. Given that the Master interacts with the user through the System Manager (DMASON Web User Interface), it must be executed in one compute node with external communication via ssh tunneling. Therefore, in all experiments, we allocate one compute node for the Master process and one additional node for each Worker process. It is worth mentioning that achieving this configuration is technically complex and requires specific settings from the HPC platform administrators. So, some works, such as [14], have had to give up the experimentation using this framework.

DMASON is a Java-based development framework, so a Java Virtual Machine (JVM) must be deployed in each compute nodes executing a simulator Worker process. We have taken advantage of the high memory capacity of Tegner's nodes (Tegner thin nodes with 512 GB) to assign up to 384 GB of memory for the heap of each JVM. It will improve the overall performance of the simulator and allows the execution of experiments with significant higher number of agents than in Beskow.

It is worth noticing that executing each process in a JVM may produce a significant variability in the experiment results because the JVM will decide how to manage the available 24 CPU cores to run the process Java threads (one for each Worker's slot), communication threads (one for each ActiveMQ's communication channel), and mechanisms such as the garbage collection. In this case, we use the mode of the execution time instead of its average to assess the simulator behavior.

Finally, Master and Worker processes will communicate using ActiveMQ, based on Java Message Service (JMS), through the platform's low-latency Infiniband EDR interconnection (up to 12 GB/s).

Tuning the number of slots per worker

As explained in Section 2.1, in a simulator generated with DMASON, the Master process sends regions of the simulated space (field) to the Worker processes, which can simulate more than one region at a time. The maximum number of regions that can be assigned to the same Worker node is limited by its configurable number of slots.

To tune the number of slots that will be offered by each Worker in a Tegner (thin) node, we have executed our base case (M-M ABMS class, 1024 × 1024 2D world, and an AoI of 10) for 1 Ki to 8 Ki agents, using one Worker process on one compute node. For each execution, the Worker process has been configured with a variable number of slots in the range [4:100].

Fig. 11 shows that the configuration with 16 slots gives the best results in all cases, especially when the number of agents increases. This is an excellent result because it means that we can set this parameter to 16 for all the experiments in the performance analysis.

This parameter depends on the threading capabilities of the JVM deployed at a compute node and the number of cores available in the node. Therefore, this analysis should be done for each computation platform used to tune this parameter.

Performance analysis

Fig. 12 shows the execution time of the M-M ABMS class for 1, 2, 4, and 8 Worker nodes, each offering 16 slots, and 1 Ki, 2 Ki, 4 Ki, and 8 Ki agents.

When examining the strong scalability analysis, it can be seen that the execution time scales well from 1 to 2 and 2 to 4 Worker nodes (from 24 to 96 cores), but the improvement is much smaller from 4 to 8 Worker nodes (96 to 192 cores). Scalability is worse for the smallest cases (1 Ki and 2 Ki) because of the higher ratio between communication and computation load.

In order to give a deeper insight into the strong scalability analysis, Table 3 shows the efficiency of each execution regarding the 1 Worker node case (time for 24 cores) using Eq. (1). The expected reduction in efficiency can be observed when increasing

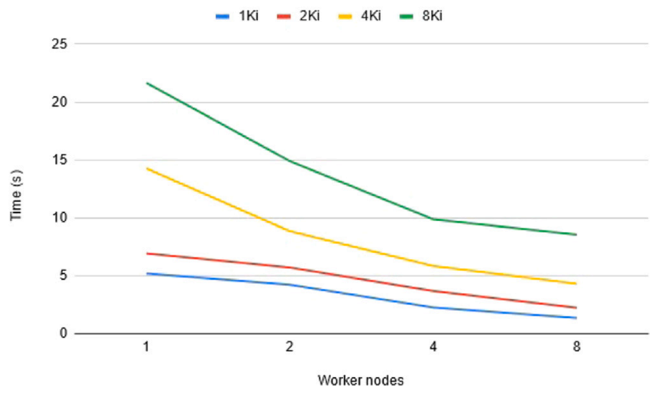


Fig. 12. Strong scalability analysis results for DMASON.

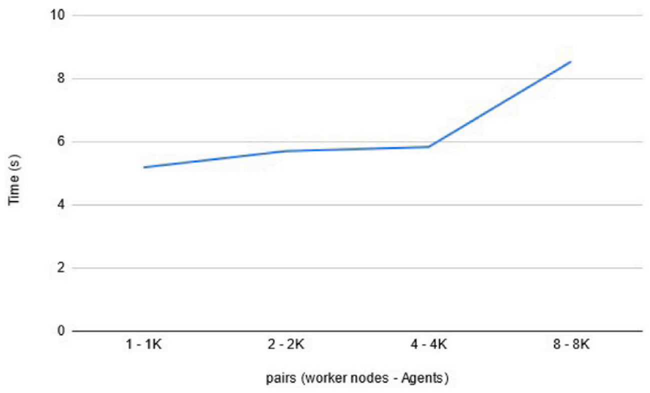


Fig. 13. Weak scalability analysis results for DMASON with (worker nodes, agents) pairs: (1, 1 Ki), (2, 2 Ki), (4, 4 Ki), (8, 8 Ki).

Table 3
Strong scalability analysis: Efficiency index.

Agents/workers	1	2	4	8
1 Ki	100	61.40	57.55	48.06
2 Ki	100	60.70	47.07	38.87
4 Ki	100	80.65	61.13	41.47
8 Ki	100	72.70	54.84	31.68

Table 4
Weak scalability analysis: Time ratio.

Agents/Workers	1	2	4	8
1 Ki,2 Ki,4 Ki,8 Ki	1	1.10	1,12	1,65

the number of resources. In fact, for the biggest cases (4 Ki and 8 Ki agents), the efficiency loss worsens for each increment in the number of resources as expected (by Amdahl’s Law).

To carry out the weak scalability analysis, we must increase the number of resources and the number of agents in the same proportion. Therefore, for this analysis, we can use Fig. 13, which shows the execution time for the following configuration pairs (worker nodes -24 cores-, number of agents): (1, 1 Ki), (2, 2 Ki), (4, 4 Ki), and (8, 8 Ki). In the ideal case, the execution time should be the same for each combination of the number of agents and resources. It can be seen that the execution time from 1 Worker node to 4 Worker nodes is almost the same, but there is a significant increase in the execution time from 4 to 8 Worker nodes.

In order to give a deeper insight into the weak scalability analysis, Table 4 shows the evolution of the ratio between the execution time of each simulation in relation to the 1 Worker node case (time for 24 cores). It is easy to see that for 8 Worker nodes, this ratio sharply increases a 65%. Although these values can be considered good enough, the last case significant increment signals a negative trend.

Finally, regarding the reliability analysis, we study the simulation results fixing the number of resources and increasing the number of agents. Fig. 14 shows the results for varying the number of agents from 1 Ki to 1024 Ki, using a fixed number of cores,

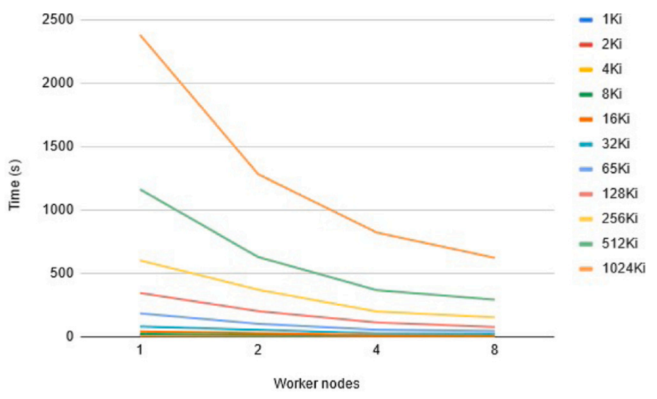


Fig. 14. Reliability analysis: DMASON using 16 slots for each worker.

Table 5
Reliability analysis: Execution time per agent (ms).

Agents/Workers	1	2	4	8
1 Ki	5,07	4,13	2,00	1,32
2 Ki	3,38	2,79	1,80	1,09
4 Ki	3,48	2,16	1,42	1,05
8 Ki	2,64	1,82	1,20	1,04
16 Ki	2,56	1,68	1,08	0,85
32 Ki	2,49	1,68	0,81	0,73
64 Ki	2,83	1,56	0,85	0,68
128 Ki	2,64	1,54	0,86	0,59
256 Ki	2,30	1,42	0,76	0,58
512 Ki	2,22	1,20	0,70	0,56
1024 Ki	2,27	1,22	0,78	0,59

and for 1, 2, 4, and 8 Worker nodes. Given that the amount of available main memory in a Tegner node is very high (512 GiB), we have been able to assign 384 GiB of main memory to each JVM and, consequently, for the executed cases, the available resources for the simulation are more than enough. Table 5 shows the average agent execution time, which decreases with the number of agents up to 1024 Ki, where this time starts to increase. These results corroborate the conclusion that the simulator is still far from suffering resource contention.

Changing the agents' computation workload

This final experiment has been designed to visualize the ABMS class dimension, which has been discarded in the previous analysis because of the complexity in configuring the execution cases in DMASON.

We have fixed the number of agents to 512 Ki and changed the computation load of each agent in order to generate results for the M-L, M-M, and M-H ABMS classes. According to the specification shown in Table 1 computation load for the M-L class is modeled by calculating an FFT on an 8B table, for the M-M class by calculating an FFT on a table of 16 KiB, and for the M-H class by calculating an FFT on a table of 128 KiB. However, the simulation using DMASON of the load corresponding to a table of 128 KiB took too much time, and consequently, we have reduced the table size to 32 KiB and 64 KiB.

Fig. 15 shows the results obtained for these computation loads using 1 to 8 Worker nodes. Considering the 8 B, 16 KiB, and 64 KiB loads, the results are expected because the execution time is increasing in the same proportion as the load. However, DMASON is producing a surprising result for the 32 KiB case because the execution time is less than 2 times the one for 16 KiB. As mentioned before, running each process on a JVM produces significant variability in the experiment's execution time, which, combined with the low number of iterations that could be run, could explain this unexpected outcome.

Considering that we are obtaining the expected outcomes along the ABMS class dimension for one case, it points out that the results obtained for analyzing the performance of DMASON on Tegner using only one class could be used to estimate the results for the rest of the classes.

5. Case study: Infection & contact tracing model

Except for a few cases (small number of cores and problem sizes with low communication volume), the simulators generated by RepastHPC have clearly overcome the ones generated by FLAME on Beskow. Considering that this evaluation has been conducted in the context of the Epi-EWS project, it seems that implementing an ABMS simulating epidemic infection spread and contact tracing on Beskow should be done using RepastHPC. To validate this conclusion, we must show that the behavior of this simulator corresponds to the characteristics taken into consideration in the proposed methodology.

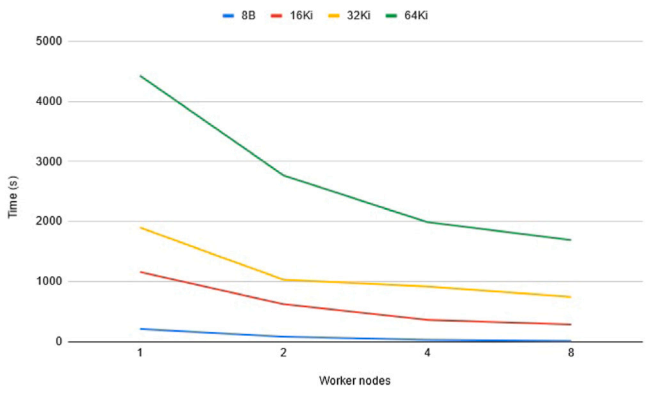


Fig. 15. Class dimension analysis: 512 Ki agents case, Worker nodes used vs. time (s) for different agents FFT computation load.

The Coronavirus disease 2019 (COVID-19) has become one of the biggest threats to humanity [35]. Epidemiological forecasting models are a good tool to design, evaluate and implement public health policies. The classical approach of equation-based models to model the spread of an infectious disease in a population has severe limitations because they assume that the population being modeled is homogeneous. The alternative is ABMS. In this case, each agent can be given different attributes and make different decisions, allowing the model to capture interactions and behavior at the individual level [36].

As presented in [37], an epidemiological agent-based model has four main components: disease, society, transportation, and environment. To model the disease, we must define how the infection is transmitted between agents and how it progresses in an infected agent. Modeling society means simulating the population, while modeling transportation determines how the agents move in the environment. Finally, modeling the environment involves creating the space in which the agents interact.

For this study, we have adapted an infection model [38] already developed using the Pandora framework [23], which simulates the infection spreading and contact tracing in the street. It should be noted that Pandora has not been contemplated in this study because it cannot be considered, in its current development stage, a stable and portable HPC ABMS development framework.

The main features of the adapted model are the following:

- **Space**. Agents move in a rectangular area representing a street. This area is modeled with a grid, and at most, one agent can be placed in each grid position.
- **Movement**. Agents travel up or down the street and, in every simulation step, an agent tries to move to the next grid position in its traveling direction. The agent can also try to move one position left or right with a certain probability. This movement will only occur if the target position is free. Otherwise, the agent rests at the original position.
- **Initial agents distribution**. The simulation begins with a given number of agents uniformly distributed over the space.
- **Birth of agents**. New agents are added randomly at both ends of the street, and always move towards the other end.
- **Death of agents**. Agents disappear when they go beyond one of the ends of the street.
- **Infection spreading**. Only sick agents can infect other agents, and infected agents do not have this capacity. At the beginning of the simulation, a given portion of agents are sick. In every simulation step, every sick agent might randomly infect agents in its area of influence.
- **Contact tracing**. Agents have a mobile application that records all agents in its area of influence and whether they were infected or not.

Upon analyzing the model, it becomes evident that, from the perspective of an individual agent, the computation and communication volume is relatively low. This is due to the fact that each agent is only required to interact with a small number of neighboring agents for tasks such as spreading and contact tracing. However, contact tracing requires a significant amount of memory (every contact is registered), and agents are constantly moving. Thus when many agents move from one computation node to another, it implies a high volume of communication. Consequently, according to Table 1, this model can be classified as an M-L or even an H-L ABMS class.

Table 6 shows the main parameters of the model developed in RepastHPC. The street is a rectangle of 200×1200 grid positions (Space dimension). Agents try to move left or right in 9% of the simulation steps (Movement drifting), the simulation begins with 3% of infected agents (Sick rate) and the probability of a sick agent infecting agents inside the area of influence (Infection radius) is 0.4% (Infectiousness). Agent initial density means the percentage of grid positions ($200 \times 1200 = 240\,000$) occupied by an agent at the beginning of the simulation.

We have conducted two experiments to validate the hypothesis that this model corresponds to either the M-L or H-L classes in our classification. First, a strong scalability analysis has been done for the case of a 10% of agents' initial density, starting with 32 cores and doubling the number of cores up to 512. Second, a weak scalability analysis has been done, starting with a 10% of agents' initial density and 32 cores doubling the initial density and number of cores up to 80% of agents' initial density and 256 cores.

Table 6
Model parameters.

Parameter	Value
Space dimension	1200 × 200
Infection radius	2
Movement drifting	9%
Agent initial density	10% 20% 40% 80%
Sick rate	3%
Infectiousness	0.4%

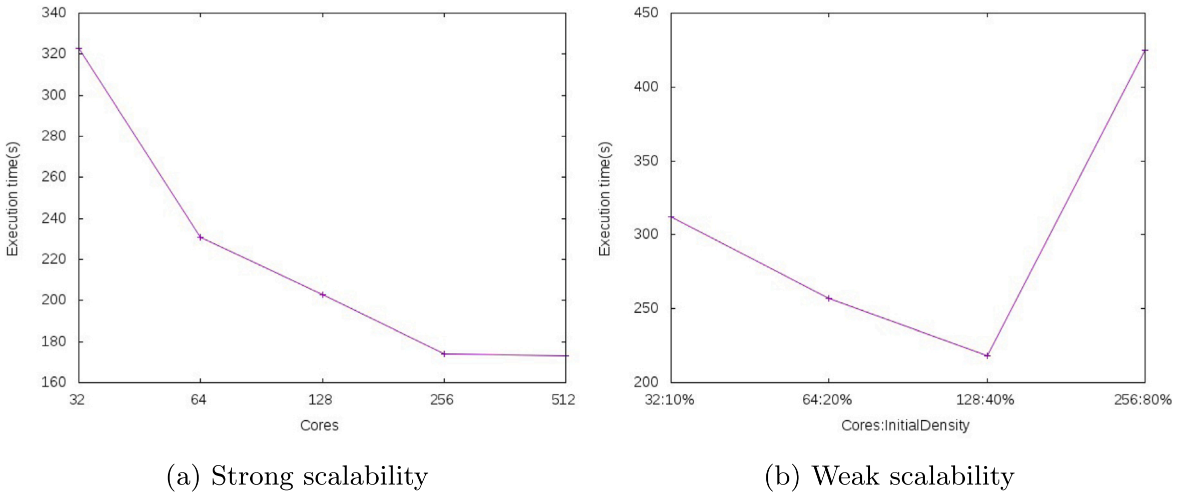


Fig. 16. Results of the infection and contact tracing model implemented in RepastHPC.

Fig. 16 shows the average execution time for 5 executions of both experiments, and Fig. 17 shows the values of Efficiency and Time ratio indexes, which are the indexes used for assessing the strong and weak scalability results, respectively.

In the strong scalability case, it can be seen that the efficiency is reducing quickly, which is following the results shown for the efficiency index in Fig. 5(c) in all the low computation load cases. The obtained results closely match those of the M-L and H-L classes. In the case of the weak scalability analysis, it can be observed that the simulation scales very well for the $64 : 20\%$ and $128 : 40\%$ executions (time ratio below 1.0) while worsening a 40% for the $256 : 80\%$ execution. These results are also following the ones shown for the time ratio in Fig. 7(c), which shows that in the low computation classes, this index starts growing earlier. In other words, the simulators generated by RepastHPC scale well but not perfectly.

Summarizing, the obtained results validate our hypothesis about the classification of this model and the decision to implement it using RepastHPC. Moreover, they also indicate that the proposed methodology can be helpful for deciding the adequate ABMS development framework for real problems.

6. Related work

Several comparative studies, such as [15,39], and [14], have been presented. In [39], five software platforms for scientific agent-based models (NetLogo, MASON, Repast, Java, and Objective-C versions of Swarm) were reviewed using test models. This study compares the execution time of the platforms in a single server and does not claim to define a formal benchmark. It also raises other issues regarding the difficulty of using each framework (e.g., programming language) or the lack of scientific tools for modeling, and establishes priorities for the continued development and improvement of ABMS platforms.

More recently, [14] presented a survey on parallel distributed multi-agent simulation frameworks. The analyzed frameworks were: DMASON, FLAME, Jade, Pandora, RepastHPC, PDES-Mas, SWAGES, Ecolab, MACE3J, and ABM++. This study qualitatively assesses these frameworks considering their main features (e.g., agent distribution and communication scheme). In addition, it presents a performance evaluation of FLAME and RepastHPC using a test example. We used the example introduced in this study as a base for developing the benchmark presented in our previous work [25] and the comprehensive evaluation methodology described in this work.

Probably the most complete review of the state-of-art frameworks for using agent-based computing technology was presented in [15]. This study covers a wide spectrum of agent-based modeling and simulation tools (85). It aimed to provide an ontological reference for classifying agent-based modeling and simulation development frameworks. It does not introduce examples or experimental results for any of the considered frameworks.

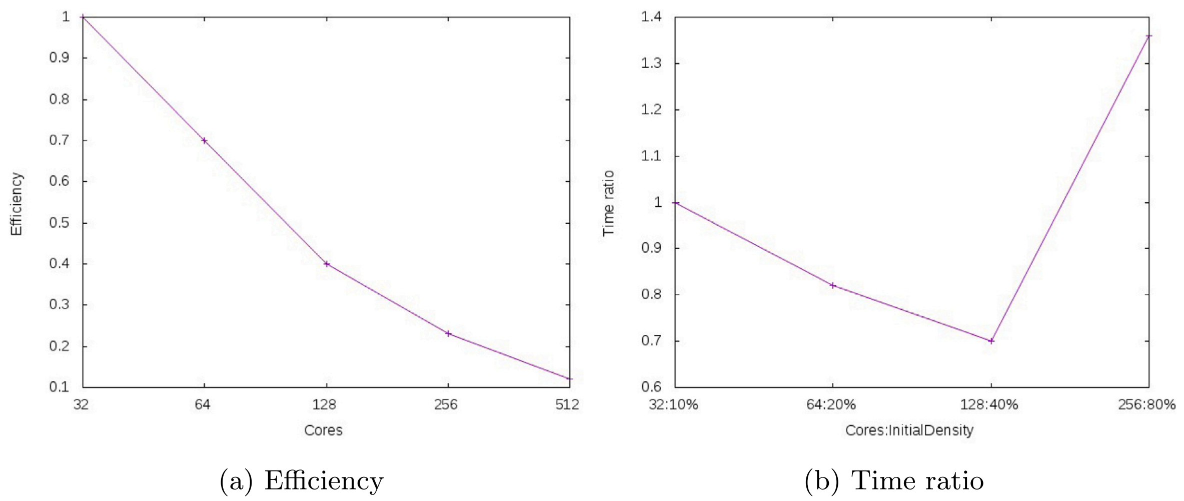


Fig. 17. Efficiency and time ratio of the infection and contact tracing model implemented in RepastHPC.

Apart from our previous contribution [25], we have found that the primary attempt to create a benchmark for assessing parallel ABMS applications performance came from the OpenAB group [40] and was published in [41,42]. The first proposal [41] implements one FLAME demo for FLAME GPU [19], MASON [22], and REPAST Simphony [43]. However, although this is a nicely formalized proposal, it is limited to a fixed radius near neighboring ABMS. The second one [42] models a set of molecules following Brownian Dynamics methods in FLAME GPU. Similarly to our benchmark proposal, this benchmark introduces the variation of the system workload during the execution. However, the model does not consider parameters such as the message size, the amount of computation, or the number of agents interacting within a certain distance.

Finally, a first approach of a validation methodology for agent-based models was presented in [44] for macroeconomic models. The validity of models is still a challenge in agent-based modeling. This paper shows a methodology based on three levels: micro, meso, and macro. At the micro level, the agent behavior should be similar to the observed agent behavior. At the meso level, model output is compared with the distributions of aggregate variables. Finally, at the macro level, correlations between aggregate variables are validated.

In summary, to the best of our knowledge, there is no existing approach that combines the characterization of different classes of ABMS with a systematical evaluation of an abstract implementation of these classes using a benchmark, and further validating the results through real executions on a specific hardware platform.

7. Discussion

The main achievement of this work is the definition of a systematical methodology for selecting the HPC ABMS development framework that could deliver the best performance for a given problem and hardware platform. This methodology is based on the definition of a set of representative problem classes, the synthetic implementation of these classes using a previously developed benchmark, and the analysis of the performance obtained for a well-defined set of configurations.

The approach proposed in the literature to assess HPC ABMS development frameworks has been focused on comparative studies considering several characteristics of these frameworks. Although most of these studies adequately address the comparison based on aspects such as the difficulty of using each framework, programming language, agent distribution, or communication scheme, they lack a comprehensive assessment of the performance of the generated simulators. Consequently, we consider that this work fills this gap and could be a valuable complement to the traditional approach because, to the best of our knowledge, it is the first time a systematical performance evaluation methodology for HPC ABMS development frameworks has been proposed.

We initially intended to test the proposed methodology for three of the most used ABMS development frameworks on HPC platforms: FLAME, RespastHPC, and DMASON. However, we found here the first barrier, services offered by an HPC platform can limit the ABMS development framework that can be used, especially those not adequately designed for HPC. For example, DMASON cannot be executed on the Beskow platform because it uses a web interface with a tunneled ssh connection to the DMASON Master process during the simulation, and this forces to manually set the parametric configuration of each experiment. Consequently, we had to test DMASON on a different platform (Tegner), which could be appropriately configured.

In the strong scalability analysis, RepastHPC outperforms FLAME in all cases, especially for the ABMS classes with the highest communication volume. The fundamental reason for this difference is the centralized communication mechanism of FLAME. Focusing on FLAME, it can also be observed that for smaller numbers of cores, Round-Robin partitioning outperforms geometric one. In contrast, for higher numbers of cores, it is the other way around. In the first case, balancing the computation load has a more significant impact on performance than the communication overhead, while in the second case, communication overhead is more important than load balancing.

The weak scalability analysis shows that FLAME scalability is far from ideal, especially for executions with a higher number of agents, because its centralized communication mechanism leads to increasing demand for memory. RepastHPC results seem closer to the ideal because it uses a distributed communication mechanism that minimizes the impact on memory usage.

The reliability analysis shows that RepastHPC may fail for a small number of agents because agents are not assigned to some cores. It happens because the generated simulator executes the simulation loop regardless of the number of assigned agents; thus developers must control this condition explicitly. FLAME may fail for a higher number of agents and the highest communication volume. In these cases, the simulator could deplete the main memory available, and consequently, the program crashes because there are no memory checks.

Finally, when introducing heavy load imbalances, even though it does not include any load balancing mechanism, RepastHPC performance is not strongly affected. At the same time, FLAME presents wide variations in the execution time, especially for geometric partitioning. It is because for geometric partitioning, the agents that have the highest probability of reproducing are concentrated in a few nodes.

Summarizing, during the simulations conducted on the Beskow platform, RepastHPC consistently outperformed FLAME across all simulation classes. However, there were instances where both frameworks exhibited similar performance. Regarding DMASON, our work has shown strong evidence supporting previous claims [14,15] about this development framework being more suited for distributed systems than HPC platforms.

We have defined a systematical methodology for achieving a comprehensive performance characterization of an ABMS HPC development framework on a given platform. This methodology is, on the one hand, based on the definition of a set of representative ABMS classes and, on the other hand, relies on a previously developed benchmark. These two aspects lead to the main limitations of our approach:

- The definition of the representative ABMS classes results from a detailed analysis of real simulation cases. However, fixing the number of classes (9) and the values of the message sizes and FFT input for each case could be a limitation. The methodology could be extended to be more flexible and accept configurable values for these parameters.
- The agents implemented in the benchmark used in the methodology can only interact with their neighbors up to a certain configurable distance. Consequently, the methodology only considers models based on geographical relationships between agents, which are not the only possible ABMS models. For example, [2] proposes an ABMS model for simulating the effect of curation algorithms in Twitter, where relationships between agents do not depend on geographical distance. Implementing these non-spatial models could be significantly different from the spatial ones for some frameworks, which could lead to significant performance variations. In this case, the benchmark could be extended to include non-spatial ABMS models, so, they can be considered in the proposed methodology.

This work has been focused on the assessment of the performance of the simulators generated by the studied frameworks. Nevertheless, other aspects can also be considered when selecting an ABMS development framework, for example, how much easy or difficult it is to deploy the platform or implement the model. However, these aspects have been appropriately addressed in other studies, while, to the best of our knowledge, it is the first time a systematical performance evaluation methodology for HPC ABMS development frameworks has been proposed.

8. Conclusions and future work

Selecting a framework, among the multiple available possibilities, for implementing a parallel ABMS depends on multiple factors. Some are qualitative, such as developers' familiarity with the implementation language or the simplicity of defining agents' behavior and interactions. However, in the end, the most relevant decision factors are the reliability and the performance gain obtained from the generated parallel simulator.

In this work, we have introduced a methodology that allows developers to determine the most promising framework for developing a specific ABMS on a given HPC hardware platform without an actual implementation of the model. This methodology leverages a benchmark developed in previous work for implementing nine predefined simulation classes based on the simulation computation load and communication volume. These classes span from low to high computation load and communication volume.

Then, a set of performance and reliability analyses, including strong and weak scalability tests, are executed on the target platform for each of the defined classes and considered frameworks, which produces, as a result, a development framework ranking for each class.

Finally, a developer of a particular ABMS only has to be able to determine to which class belongs its model to know which development framework could lead to the best performance.

We have used this methodology to rank two of the best-known ABMS development frameworks (FLAME and RepastHPC) on Beskow (position 485 in the November 2022 TOP500 list). In addition, we also applied the methodology for analyzing the capabilities of DMASON, another very well-known ABMS development framework, on a different machine (Tegner).

Finally, to validate the proposed methodology, we implemented a real use case on this system using RepastHPC. The use case consisted of an epidemiological and contact tracing simulation of people strolling in the street. By its characteristics, this model was classified in the low computation load - medium or high communication volume class. Experimentation showed that the results obtained from the execution of this model for different cases agreed with those obtained for the corresponding class during the framework evaluation.

In future work, we expect to use our methodology with other well-known frameworks such as Jade [45] and, especially, Distributed MASON, which is a redesign of DMASON for HPC platforms [46]. In addition, we plan to extend the benchmark to generate synthetic models with non-spatial communication patterns and enrich the set of representative ABMS classes for the frameworks, such as RepastHPC, that implement spatial and non-spatial communications differently. Also related to the set of representative ABMS classes, we want to extend the methodology to be more flexible and accept configurable values in the number of classes and the parameters that define the communication and computation loads. We will also analyze the performance of ABMS models that use Geographic Information Systems (GIS) supported by some HPC ABMS frameworks. Finally, we will explore the application of HPC ABMS simulation of IoT systems, such as Smart Cities [47] and Connected Car (with V2X communication protocols) [48], to determine their characteristics and, if necessary, adapt the benchmark and the considered representative ABMS classes.

Data availability

Code is open source. Links is provided in manuscript.

Acknowledgments

This work has been supported by the Ministerio de Ciencia e Innovación MCIN AEI/10.13039/501100011033, Spain under contract PID2020-113614RB-C21 and by the Catalan government under contract 2021 SGR 00574. We acknowledge PRACE for awarding us access to Beskow and Tegner hosted by the PDC Center for High Performance Computing, KTH Royal Institute of Technology, Sweden.

References

[1] Charles M. Macal, Michael J. North, Agent-based modeling and simulation, in: Proceedings of the 2009 Winter Simulation Conference, WSC, 2009, pp. 86–98.

[2] Anna Gausen, Wayne Luk, Ce Guo, Using agent-based modelling to evaluate the impact of algorithmic curation on social media, J. Data Inf. Qual. (2022).

[3] Emilio Serrano, Carlos A. Iglesias, Validating viral marketing strategies in Twitter via agent-based social simulation, Expert Syst. Appl. 50 (2016) 140–150.

[4] Yadong Xu, Wentong Cai, Heiko Aydt, Michael Lees, Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic simulation, in: Proceedings of the Winter Simulation Conference 2014, 2014, pp. 3483–3494.

[5] Li Qiang, Xuefeng Guan, Rui Li, Huayi Wu, 4D-SAS: A distributed dynamic-data driven simulation and analysis system for massive spatial agent-based modeling, ISPRS Int. J. Geo-Inf. 5 (2016) 42.

[6] Emily Berglund, Using agent-based modeling for water resources planning and management, J. Water Resour. Plan. Manag. 141 (2015) 04015025.

[7] Sonja Kolen, Stefan Dähling, Timo Isermann, A. Monti, Enabling the analysis of emergent behavior in future electrical distribution systems using agent-based modeling and simulation, Complexity 2018 (2018) 1–16.

[8] Shu-Heng Chen, Chia-Ling Chang, Ye-Rong Du, Agent-based economic models and econometrics, Knowl. Eng. Rev. 27 (2012) 187–219.

[9] Bo Zhang, Donald L. DeAngelis, An overview of agent-based models in plant biology and ecology, Ann. Botany 126 (4, SI) (2020) 539–557.

[10] S. Rikard, Thomas Athey, Anders Nelson, Steven Christiansen, Jia-Jye Lee, Jeffrey Holmes, Shayn Peirce, Jeffrey Saucerman, Multiscale coupling of an agent-based model of tissue fibrosis and a logic-based model of intracellular signaling, Front. Physiol. 10 (2019) 1481.

[11] Claudio Savaglio, Maria Ganzha, Marcin Paprzycki, Costin Bădică, Mirjana Ivanović, Giancarlo Fortino, Agent-based Internet of Things: State-of-the-art and research challenges, Future Gener. Comput. Syst. 102 (2020) 1038–1053.

[12] Peng Jing, Hanbin Hu, Fengping Zhan, Yuexia Chen, Yuji Shi, Agent-based simulation of autonomous vehicles: A systematic literature review, IEEE Access 8 (2020) 79089–79103.

[13] Constantin-Valentin Pal, Florin Leon, Marcin Paprzycki, Maria Ganzha, A review of platforms for the development of agent systems, 2020, ArXiv, abs/2007.08961.

[14] Alban Rousset, Bénédicte Herrmann, Christophe Lang, Laurent Philippe, A survey on parallel and distributed multi-agent systems for high performance computing simulations, Comp. Sci. Rev. 22 (2016) 27–46.

[15] Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, Gregory M.P. O'Hare, Agent Based Modelling and Simulation tools: A review of the state-of-art software, Comp. Sci. Rev. 24 (2017) 13–33.

[16] Christophe Deissenberg, Sander van der Hoog, Herbert Dawid, EURACE: A massively parallel agent-based model of the European economy, Appl. Math. Comput. 204 (2) (2008) 541–552.

[17] Jonathan Ozik, Justin M. Wozniak, Nicholson T. Collier, Charles M. Macal, Mickaël Binois, A population data-driven workflow for COVID-19 modeling and learning, Int. J. High Perform. Comput. Appl. 35 (5) (2021).

[18] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn Chin, David Worth, Chris Greenough, Exploitation of high performance computing in the FLAME agent-based simulation framework, in: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems, 2012, pp. 538–545.

[19] Paul Richmond, Simon Coakley, Daniela M. Romano, A high performance agent based modelling framework on graphics card hardware with CUDA, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, Vol. 2, AAMAS, 2009, pp. 1125–1126.

[20] Repast Project, The repast suite. https://repast.github.io/.

[21] Russell K. Standish, Going stupid with EcoLab, Simulation 84 (12) (2008) 611–618.

[22] Gennaro Cordasco, Rosario De Chiara, Ada Mancuso, Dario Mazzeo, Vittorio Scarano, Carmine Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: the experience with D-MASON, Simulation 89 (10) (2013) 1236–1253.

[23] Xavier Rubio-Campillo, Pandora: A versatile agent-based modelling platform for social simulation, in: Proceedings of SIMUL 2014, the Sixth International Conference on Advances in System Simulation, 2014, pp. 29–34.

[24] Francisco Borges, Albert Gutierrez-Milla, Emilio Luque, Remo Suppi, Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling, Future Gener. Comput. Syst. 68 (2017) 59–73.

[25] Andreu Moreno, Juan J. Rodríguez, Daniel Beltrán, Anna Sikora, Josep Jorba, Eduardo César, Designing a benchmark for the performance evaluation of agent-based simulation applications on HPC, J. Supercomput. 75 (2019) 1524–1550.

[26] MPI Forum, Message Passing Interface. http://mpi-forum.org.

[27] Ivan Trenčanský, Radovan Červenka, Agent Modeling Language (AML): A comprehensive approach to modeling MAS, Informatica (Slovenia) 29 (2005) 391–400.
[28] M. Frigo, S.G. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) 216–231.
[29] Sameer Shende, TAU performance system, in: IWOCL'22: International Workshop on OpenCL, Bristol, United Kingdom, May 10 - 12, 2022, ACM, 2022, p. 12:1.
[30] Evolutionary Computation Laboratory and Center for Social Complexity, George Mason University and ISISLab, University of Salerno, Distributed MASON. https://cs.gmu.edu/~eclab/projects/mason/extensions/distributed/.
[31] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, Gabriel Balan, Mason: A multiagent simulation environment, Simulation 81 (7) (2005) 517–527.
[32] Charles Macal, Michael North, Introductory tutorial: Agent-based modeling and simulation, in: Proceedings of the Winter Simulation Conference 2014, 2014, pp. 6–20.
[33] Guiyeom Kang, Claudio Márquez, Ana Barat, Annette T. Byrne, Jochen H.M. Prehn, Joan Sorribes, Eduardo César, Colorectal tumour simulation using agent based modelling and high performance computing, Future Gener. Comput. Syst. 67 (2017) 397–408.
[34] PDC Center for High Performance Computing, Beskow CRAY XC40. https://www.pdc.kth.se/about/history-of-pdc/recent-systems-at-pd/beskow-1.737436.
[35] Alexander E. Gorbalenya, Susan C. Baker, Ralph S. Baric, Raoul J. de Groot, Christian Drosten, Anastasia A. Gulyaeva, Bart L. Haagmans, Chris Lauber, Andrey M. Leontovich, Benjamin W. Neuman, Dmitry Penzar, Stanley Perlman, Leo L.M. Poon, Dmitry V. Samborskiy, Igor A. Sidorov, Isabel Sola, John Ziebuhr, Coronaviridae Study Group of the International Committee on Taxonomy of Viruses, The species Severe acute respiratory syndrome-related coronavirus: classifying 2019-nCoV and naming it SARS-CoV-2, Nat. Microbiol. 5 (4) (2020) 536–544.
[36] Georgiy V. Bobashev, D. Michael Goedecke, Feng Yu, Joshua M. Epstein, A hybrid epidemic model: Combining the advantages of agent-based and equation-based approaches, in: 2007 Winter Simulation Conference, 2007, pp. 1532–1537.
[37] Elizabeth Hunter, Brian Mac Namee, John D. Kelleher, A taxonomy for agent-based models in human infectious disease epidemiology, J. Artif. Soc. Soc. Simul. 20 (3) (2017) 2.
[38] Xavier Rubio, pandora/examples/epidemy. https://github.com/xrubio/pandora.
[39] Steven F. Railsback, Steven L. Lytinen, Stephen K. Jackson, Agent-based simulation platforms: Review and development recommendations, Simulation 82 (9) (2006) 609–623.
[40] Open agent benchmark initiative for parallel and distributed benchmarking. http://www.openab.org.
[41] R. Chisholm, P. Richmond, S. Maddock, A standardised benchmark for assessing the performance of fixed radius near neighbour, in: Euro-Par 2016: Parallel Processing Workshops, 2016, pp. 311–321.
[42] E. Alzahrani, P. Richmond, A.J.H. Simons, A formula-driven scalable benchmark model for ABM, applied to FLAME GPU, in: Euro-Par 2017: Parallel Processing Workshops, 2017, pp. 703–714.
[43] Nicholson Collier, Michael North, Parallel agent-based simulation with repast for high performance computing, Simulation 89 (10) (2013) 1215–1235.
[44] Tieleman Sebastiaan, Towards a validation methodology for macroeconomic agent-based models, Comput. Econ. (2021).
[45] Nikolaos Spanoudakis, Pavlos Moraitis, Modular JADE agents design and implementation using ASEME, in: 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Vol. 2, 2010, pp. 221–228.
[46] Haoliang Wang, Ermo Wei, Robert Simon, Sean Luke, Andrew Crooks, David Freelan, Carmine Spagnuolo, Scalability in the MASON multi-agent simulation system, in: E. Besada, O.R. Polo, R. Degrande, J.L. Risco (Eds.), Proceedings of the 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2018, pp. 135–144.
[47] Sunny Prakash Prajapati, Rahul Bhaumik, Tarun Kumar, An intelligent ABM-based framework for developing pandemic-resilient urban spaces in post-COVID smart cities, Procedia Comput. Sci. 218 (2023) 2299–2308, International Conference on Machine Learning and Data Engineering.
[48] Vicente R. Tomás, Luis A. García, Adrian León Alonso, An agent-based platform to evaluate V2X routing road traffic scenarios, Simul. Model. Pract. Theory 125 (2023) 102750.