# Nested Graphs: a model to efficiently distribute multi-agent systems on HPC clusters

Alban ROUSSET, Bénédicte HERRMANN, Christophe LANG,
Laurent PHILIPPE, Hadrien BRIDE
email: firstname.name@femto-st.fr

Femto-ST Institute, Univ. Bourgogne Franche-Comté/CNRS Besançon - France
2017

**Abstract**

Computational simulation is becoming increasingly important in numerous research fields. Depending on the modeled system, several methods such as differential equations or Monte-Carlo simulations may be used to represent the system behavior. The amount of computation and memory needed to run a simulation depends on its size and precision and large simulations usually lead to long runs thus requiring to adapt the model to a parallel system. Complex systems are often simulated using Multi-agent systems (MAS). While linear system based models benefit from a large set of tools to take advantage of parallel resources, multi-agent systems suffer from a lack of platforms that ease the use of such resources. In this paper, we propose the use of Nested Graphs for a new modeling approach that allows the design of large, complex and multi-scale multi-agent models which can efficiently be distributed on parallel resources. Nested Graphs are formally defined and are illustrated on the well-known predator-prey model. We also introduce PDMAS (Parallel and Distributed Multi-Agent System) a platform that implements the Nested Graph modeling approach to ease the distribution of multi-agent models on High Performance Computing clusters. Performance results are presented to validate the efficiency of the resulting models.

**keywords**

Multi-Agent Simulation, Parallel, Nested Graph, High Performance Computing

## 1 Introduction

Computational simulation is becoming increasingly important in numerous research fields like psychology [7] or biology [30] but even in fields that do not traditionally use computational models such as archaeology and anthropology. Depending on the characteristics of the modeled system, several methods such as differential equations or Monte-Carlo simulations may be used to represent the system behavior. Multi-agent systems (MAS) are often used to model and simulate complex systems. In such systems the complexity of the dependencies between the phenomena that drive the entities behavior makes it difficult to define a global law that models the entire system. Based on a simple algorithmic description of individual behaviors, multi-agent systems provide a support to observe global developments emerging from a set of interacting agents. Recently, the interest for parallel multi-agent platforms has increased as parallel platforms offer more resources to run larger agent simulations and thus allow to obtain results previously intractable using a smaller number of agents (e.g. simulation of individual motions in a city/urban mobility).

Whatever the modeling approach used, increasing the size and the precision of a model increases the amount of computation and the size of memory used. More computing resources are thus needed and centralized systems are often no longer sufficient to run these simulations, requiring the use of parallel systems to avoid too long runs and to provide enough memory. While linear systems

based models benefit from a large set of parallel libraries to take advantage of many computing nodes and run large simulations, multi-agent systems suffer from a lack of platforms that ease the use of parallel systems. From the viewpoint of increasing the size or accuracy of simulations, multi-agent systems are constrained to the same rules as other modeling techniques. Only some simple models may benefit from the approach used to parallelize linear systems and there is few generic approaches available to efficiently run more general agent models on parallel machines such as clusters. Due to less regular interactions and more dynamic behaviors of agents compared to linear systems, parallelizing an agent-based model is usually difficult. The parallelization difficulty, however, depends on the modeling technique and the associated data structures. So we can raise the question: do traditional ways to model MAS fit the parallelization step? Our aim is then to propose a simple way to model Parallel and Distributed Multi-Agent Systems (PDMAS) in a manner that allows them to be efficiently distributed and executed on parallel systems.

The contribution of this paper is to propose the use of Nested Graphs (NG) [28, 34] as a data structure to represent agent models and parallelize them more easily. We show that this data structure provides an elegant and efficient solution to distribute and parallelize simulations at different levels. To demonstrate and validate the advantages of using Nested Graph data structures we have implemented several models and assessed their performance on HPC resources. This paper extends and develops the results of a previous paper [34] mainly with more explanations on the modeling approach and experiment results, including a new model.

This article is organized as follows. In section 2, we give an overview and related work on the Multi-Agents Systems (MAS) and the Parallel and Distributed Multi-Agent platforms (PDMAS) context before identifying the limits regarding distribution in existing platforms. In section 3, we present our proposal of using Nested Graphs (NG) to model MAS, we illustrate its use on the well-known prey-predator model in section 4 and we explain the advantages of using NG structures to distribute a model in section 5. We present some execution results of our method in section 6. Finally, we present our conclusion and future work.

## 2 Context and related work

The concept of agent has been studied extensively for several years and in different domains. One of the first definitions of the agent concept is due to Ferber [17] :

> "An agent is a real or virtual autonomous entity, operating in a environment, able to perceive and act on it, which can communicate with other agents, which exhibits an independent behavior, which can be seen as the consequence of his knowledge, its interactions with other agents and goals it needs to achieve".

A multi-agent system, or MAS, is a platform which provides support to run simulations based on several autonomous agents. These platforms implement functions that provide services such as agent lifecycle management, communication between agents, agent perception or environment management. Among the most known platforms we can cite NetLogo [36], MadKit [19], Mason [26] and Gama [35]. There exist several papers that propose a survey on these multi-agent platforms [37, 5, 2, 21]. These platforms are designed to run on only one computer and they do not natively implement a support to run models in parallel. For large models, the memory space and computation power of only one computer are sometimes no longer sufficient to run the model. For example, this is the case if we want to simulate the individual behavior of urban mobility [10] in a large city. Increasing the size or the precision of models could, however, bring emergent behaviors that we never expected or would never seen otherwise. Using parallel systems is a way to overcome these limits in terms of computation power and memory space.

Possible approaches to distribute or parallelize a simulation include the development of a dedicated model as in [38] or the implementation of a wrapper over an existing platform [3]. These approaches are however complex as they require parallel programming skills while most of multi-agent models are developed by non-specialist programmers. Parallel and Distributed Multi-Agent Platforms (PDMAS) exist that facilitate the implementation of parallel models. We

can cite RepastHPC [12], D-Mason [14], Pandora [1] and Flame [11]. All these platforms provide a native support for parallel execution of Multi-Agent models but also important mechanisms of distribution, migration and load balancing for the simulation run. In [33] we survey existing PDMAS and compare them with a qualitative analysis and a performance evaluation.

Different types of platforms can be used to run parallel systems. Among them we can cite shared memory systems, Graphical Processing Units (GPU) or Many-Core processors (e.g Xeon Phi) and HPC clusters. While shared memory systems are easier to program and benefit from efficient tools (e.g. OpenMP) they usually provide a limited number of cores and memory space which limits simulation scalability. GPU cards and Many-Core processors are known for their efficiency on regular problems and have already been used for multi-agent simulations [25]. They however require specific development skills and their SIMD programming model only fits simple agent behaviors with as little as possible conditions. For these reasons we concentrate our work on HPC clusters.

Several key points must, however, be enforced for an efficient parallel execution of a multi-agent simulation on a HPC cluster: load balancing, agent migration, communication between agents and coherency in agent vision to cite some of them. We highlight in the following the issues raised by these key points

The load, between the processors which participate to the execution, must to be as balanced as possible in order to minimize the time spent by a processor waiting for the others. Multi-agent simulations are indeed synchronous simulations that are driven by time steps. To maintain the overall consistency of the simulation, all the processors must run in the same time step and synchronization barriers are necessary at the end of each time step. At each synchronization barrier, the processors wait until the overall synchronization is performed. In parallel multi-agent simulations, the set of agents is distributed among the processors, each running at its own speed. Running one-time step may thus take more or less time depending on the number of agents assigned to a processor and of the speed of this processor. Hence the simulation step is bounded by the slowest processor involved in the simulation and the load of the processors must then be as uniform as possible all along the simulation to be more efficient and improve running time performance. Note that balancing time steps is not specific to multi agent simulations but is also needed for instance in synchronous iterative resolutions in linear algebra.

As previously stated, agents operate in an environment. Parallelizing a multi-agent simulation thus not only implies to distribute the agents but also the environment. In numerous multi-agent simulations, agents are situated and mobile: they have a position in the environment and they can move on the environment. When the environment is distributed among computing nodes, agents must be able to migrate between environment parts. Agent migration may impact both load balancing and communication between agents. Migration may interfere with load balancing as agent migrations may generate imbalances. When an agent migrates, the system must follow the agent places in order to deliver messages [32].

Agents are able to perceive and to act on their environment. The perception field of an agent is usually limited to its neighborhood. In parallel simulations, due to the environment distribution, different parts of the simulation are run on several processors so that the perception field of an agent could be cut between different nodes, i.e. parts of the perception field are mapped on different nodes. This can be managed, by hand, in the model implementation but its leads to complex algorithmic developments. For this reason, PDMAS usually tackle this issue by providing parallelized structures. These structures provide overlapping areas, parts of the enviornment situated on the distribution borders are replicated on the neighbor nodes, to limit communications when an agent accesses the remote parts of its perception field.

Currently, the most used structure for environments in PDMAS is the grid. Grids are a good base to represent a two-dimensional environment on which agents can move and evolve. To distribute the model, most platforms use a Cartesian decomposition of the grid as presented on Figure 1.

The Cartesian decomposition allows the distribution on one axis $x$ or on two axes $x$ and $y$ as shown in Figures 2, 3. The problem is, even with a fine grain division of the grid as it is done in the D-MASON platform [6], the decomposition structure may not be flexible enough to correctly

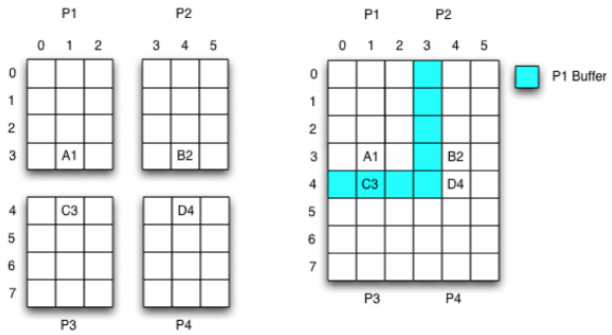



Figure 1: An example of grid decomposition for the RepastHPC platform on four processors with overlapping zone of size 1 [13]

balance the load. This is, for instance, the case when most agents are grouped in one part of the environment as a regular decomposition of the environment generates unbalanced areas in terms of number of agents. In that case, the division grain should be different depending on the agent density but such variable divisions are however very complex to implement. Moreover, square or rectangle distributions may not be appropriate for models as social models based on interaction networks [20] that do not rely on a regular structure or environment.

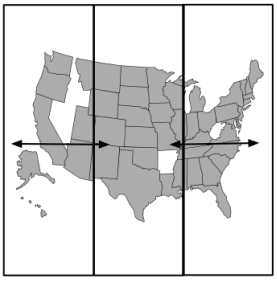


Figure 2: An example of grid decomposition on $x$ axis for the platform D-Mason on three processors [13]

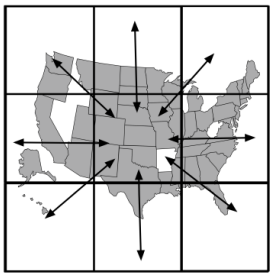


Figure 3: An example of grid decomposition on $x$ and $y$ axes for the platform D-Mason on nine processors [13]

To address this issue, we propose to use graphs as a base to model multi-agent systems. Graphs are extensively used in parallel and distributed computing as a structure to model problems such as task graphs in scheduling problem [24] or partitioning problem [22]. There are several available tools that support graph partitioning on parallel or distributed computers. We can cite Parmetis [23], PT-Scotch [9] or Zoltan [15]. These tools have great performance results even for large graphs [29]. Taking advantage of these tools is thus a way to facilitate the design of models on parallel computers and improve the execution of PDMAS. This, however, requires changing the way agent models are represented.

# 3 Proposal

A multi-agent simulation can be seen as a set of entities which interact together and with an environment, which could, in turn, be a set of agents as in [35]. Multi-agent models can thus be modeled using graphs, with vertices representing agents, and edges representing interactions or links between agents. The modeling job may, however, be a hard task because the graph structure is very likely to be complex and modeling a complex graph is not always easy or natural for MAS users. On the other hand, if the agent simulation is represented by a graph, it is more easy to

propose algorithms or tools to distribute or balance the simulation load. For these reasons, we propose the use of Nested Graphs (NG) and Nested Graph transformation as a way to model large, complex and multi-scale multi-agent simulations. This structure natively integrates a more easy-to-use and flexible graph structuring that enhances the support of distribution and load balancing on parallel platforms.

Nested Graphs are graphs where nodes can be Nested Graphs. They are recursive structures. The interesting concept introduced by Nested Graphs is the definition of different abstraction levels that can be used to conceptually divide a model in a hierarchical structure. In [28], the authors introduce a model of Nested Graphs to represent and to manipulate complex objects that they apply to databases. In the context of Multi-Agent Systems, Nested Graphs have been already used but not in a parallel and distributed context. In [27], authors use Nested Graphs to model simulations of complex systems. Using a hierarchical structure is indeed a way to more easily conceptualize the complex system that is to be modeled. We show in the following that Nested Graphs allow the description of any multi-agent system at different levels of abstraction and natively satisfy the requirements to be efficiently distributed on a parallel platform.

## 3.1 Formal description

We base our proposal on the two formal principles for a multi-scale simulation explained in [27]:

- Any agent may dynamically encapsulate an environment. This is the basis of a recursive nested structure, but this structure must be able to change in time.
- Any agent may be situated in several environments at the same time, without a prior idea of what those environments represent (a micro/macro level of the physical world, a group, an organization, a spatial memory, a social network, etc.).

We only change the first principle to adapt these definitions to our proposal of recursive structure: "Any agent may dynamically encapsulate an **agent**. This is the basis of a recursive nested structure, but this structure must be able to change in time".

In our proposal, we consider that all the components participating to the simulation are agents as in [35]. It means that the environment is modeled by one or more agents, depending on the type of environment, similar to non-environmental agents. Due to the first principle, the whole model represents a hierarchical structure of agents which provides a multi-scale mode. We then consider that each agent in the simulation is a typed and labeled Nested Graph, called Agent Graph. Agent behaviors are represented as transformations of Agent Graphs, that is to say, arcs or vertices modification in the graph. Relations between agents of the same context, i.e. relations between Agent Graphs which are contained in the same Agent Graph, are represented by typed and valued edges. These relations can, for instance, be a communication between agents or an agent position relative to an environment cell.

Formally, let $\Gamma$ be a set of types, $\Sigma$ be a set of labels and $\Lambda$ a set of values. Then the set $\mathbb{G}$ of Agent Graphs $\mathcal{G}$ is recursively defined by :

$$\mathcal{G} \in \mathbb{G} \Leftrightarrow \mathcal{G} = \langle \mathrm{G}, \mathrm{E}, \mathrm{T}, \mathrm{L}, \mathrm{V} \rangle \tag{1}$$

where $\mathrm{G} \subseteq \mathbb{G}$ is a set of Agent Graphs (also called vertices) representing the agents of the model, $\mathrm{E} \subseteq \mathrm{G} \times \mathrm{G}$ is a set of directed edges representing relations between agents, $\mathrm{T} : \mathrm{G} \mapsto \Gamma$ is a typing function assigning a type to each agent, $\mathrm{L} : \mathrm{G} \mapsto \Sigma$ is a labeling function assigning a label to each agent, and $\mathrm{V} : \mathrm{E} \mapsto \Lambda$ is a valuing function assigning a value to any edge. The state of a simulation is fully described by its Agent Graphs (equation 1).

Agent behaviors are described as Agent Graph transformations which are modeled using a pair of Agent Graphs with no labeling function (L) and a special Agent Graph containing a special node. The special node is the node executed to realize the transformation. An Agent Graph transformation can be applied to an agent of the same type as its special Agent Graph. When applied, if the special Agent Graph can be found as a sub Agent Graph of the Agent Graph of the

simulation, then the found sub Agent Graph is transformed into the second Agent Graph. The special Agent Graph of the Agent Graph transformation can be seen as a pattern which have to be recognized before being transformed into the second Agent Graph. In another word, we can see an Agent Graph transformation as a conditional structure (*if* [we have this configuration] *then* [we need to arrived to this configuration] *end*) so that the expected behavior can be performed. When a type of agent has multiple behaviors, its Agent Graph transformations are applied according to a defined workflow as usually in multi-agent system simulations.

Based on these definitions, our proposal relies on on two main points:

- A modeling method where multi-agent systems are modeled using Agent Graphs. In this method, all elements of the multi-agent model are agents without difference between environment and agents. Agents and their relations are represented by Agent Graphs. Agent behaviors are implemented based on Agent Graph transformations.
- An adapted PDMAS using Agent Graphs models is used to efficiently run the simulation on parallel platforms (see Section 5).

With this method, complex systems and complex behaviors can be modeled in a graphical way while providing a fully formal basis similar to Petri Nets [8]. It also encompasses multiple levels of abstractions needed by the modelers. The proposal is illustrated in the following section.

### 3.2 Graphical notation

We introduce here some graphical notations. Agent Graphs nodes (Figure 4) are represented by ellipses with labels of the form *Label:Type* (L,T) giving the label and type of the associated vertice. Edges between nodes (E) are represented by arrows with labels of the form *Type:Value* (T, V) giving the type and value of the associated edge.

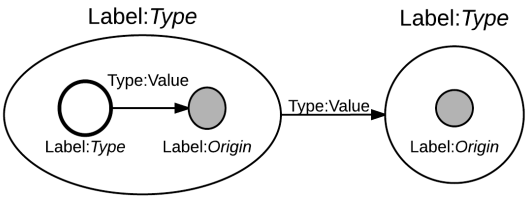



Figure 4: Generic representation of an Agent Graphs

An Agent Graph transformation is described by two Agent Graphs linked by a large arrow as shown in Figure 5 . We denote by a bold ellipse the special node of an Agent Graph transformation. In other words, the special node represents the node concerned by the transformation and which is executed.

## 4 Modeling method illustration

In this section, we illustrate the modeling method with two models the Wolf-Sheep Predation Model (WSP Model) [39] and the Virus model [40] which are classical Multi-Agent models. We choose these models as they include several representative patterns that can be found in numerous models, as mobile agents, distant interactions, concurrency between agents, etc.

### 4.1 Wolf-Sheep Predation Model

The environment of the Wolf-Sheep Predation Model is a grid composed of cells. It is the first and highest level of abstraction of our representation. It is modeled using an Agent Graph in

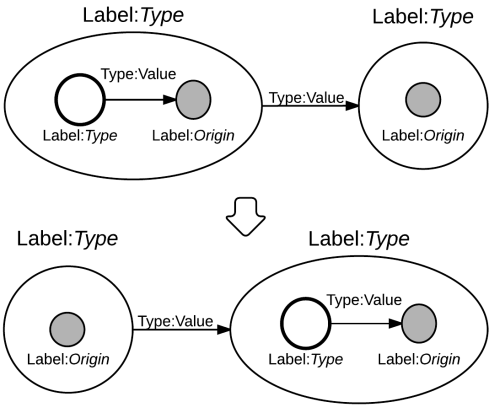



Figure 5: Generic representation of an Agent Graphs transformation

which vertices are Agent Graphs of type *Cell* and adjacent vertices are connected by arcs of type *Adjacent* (see Figure 6). Any Agent Graph of type *Cell* has a vertex of type *Origin* which is used to connect the agents contained in the same cell.

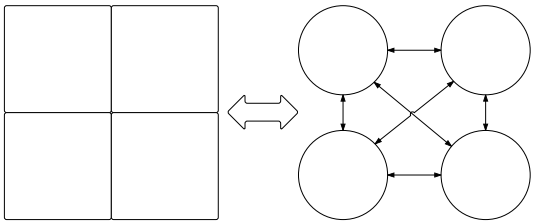


Figure 6: Graph representation of a grid

Cells contain grass, sheep and wolves. They represent a second level of abstraction. Within Agent Graphs of type *Cell*, Agent Graphs of type *Grass*, *Sheep*, and *Wolf* represent the agents present in a cell and are linked by edges of type *On* to the *Origin* vertices of the cell.

The third level of abstraction considers characteristics of agents within cells. Any Agent Graph of type *Sheep* and *Wolf* also has a vertex of type *Origin* as well as a real valued edge (from *Origin* to *Origin*) of type *Force* which represents the vital force of the considered agent. Agent Graphs of type *Grass* have no characteristic and are therefore empty.

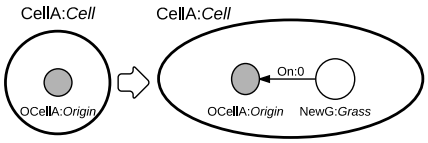
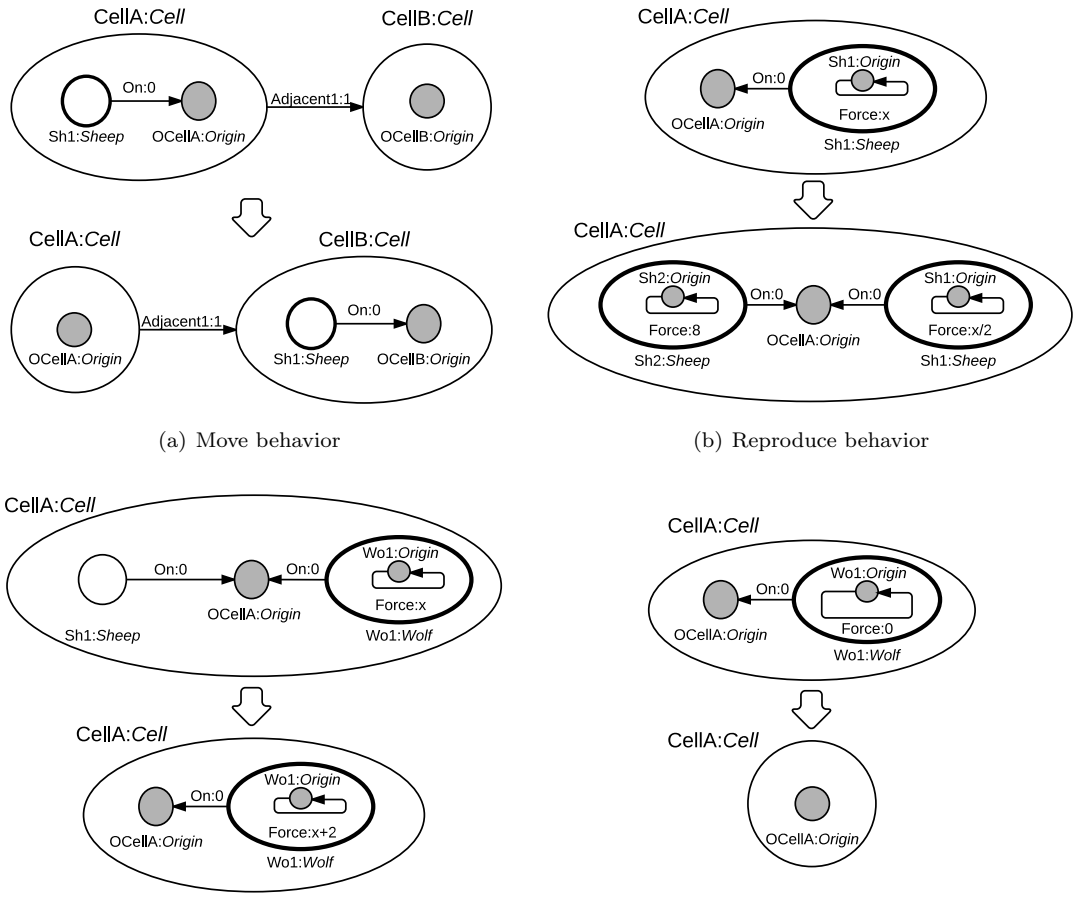



Figure 7: Grass growth behavior representation

At each time-step of the simulation, the following behaviors modeled as Agent Graph transformations are applied:

- Grass growth (Figure 7): according to a given probability grass appears on a cell.
- Move behavior (Figure 8(a)): sheep and wolves randomly move to an adjacent cell.

- Eat behavior (Figure 8(c)): sheep (resp. wolves) eat grass (resp. sheep) and increase their vital force. The grass (resp. sheep) eaten must be removed from the simulation.
- Reproduce behavior (Figure 8(b)): according to a given probability, sheep (resp. wolves) reproduce and create new sheep (resp. wolves) with a default vital force, their vital force is then divided by two.
- Die behavior (Figure 8(d)): sheep (resp. wolves) die whenever their vital force drops to zero. They must then be removed from the simulation.




(a) Move behavior

(b) Reproduce behavior

(c) Eat behavior

(d) Die behavior

Figure 8: Behavior representations for sheep and wolf agents

Figure 9 shows a workflow defining the order in which the behaviors of sheep agents are applied at each time-step. On the figure, the yellow diamonds stand for conditions and the labels on the arrows are condition results, e.g. $alea > reproductionTx$ means that a randomly generated value is greater than the reproduction rate Tx. This workflow is almost the same for wolf agents except for the eat behavior as wolf agents do not eat grass but sheep.

## 4.2 Virus Model

The virus model intends to reproduce the spread of a disease when part of a population is vaccinated. Like in the Wolf-Sheep Predation model, the environment of the Virus model is represented

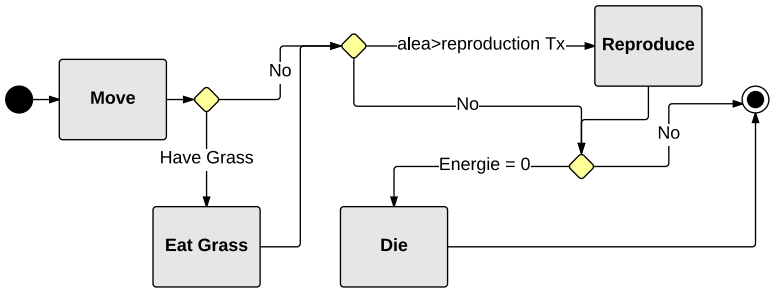



Figure 9: Diagram state transition of a sheep agent behavior

by a grid composed of cells. In the model, the cells represent the first and highest level of abstraction of the simulation. The cells contain persons which represent a second level of abstraction. Within Agent Graphs of type *Cell*, Agent Graphs of type *Person* represent the agents present in a cell and are linked by edges of type *On* to the *Origin* vertices of the cell (for example see Figure 10(a)).

Like in the Wolf-Sheep Predation model, the third level of abstraction considers characteristics of agents within cells. Any Agent Graph of type *Person* also has a vertex of type *Origin* as well as real valued edges (from *Origin* to *Origin*) of type *Force* (see Figure 10(b) for example) and *Infected* (see Figure 10(c) for example) which respectively represent the vital force and the state of the considered agent regarding infection.

At each time-step of the simulation, the following behaviors modeled as Agent Graph transformations are applied:

- Move behavior (Figure 10(a)): persons randomly move to an adjacent cell.
- Infectious behavior (Figure 10(c)): according to a given probability, a contaminated person can infect other persons in his neighborhood.
- Reproduce behavior (Figure 10(b)): according to a given probability, persons reproduce and create new persons.
- Die behavior (Figure 10(d)): persons die whenever their vital force drops to zero. They must then be removed from the simulation.

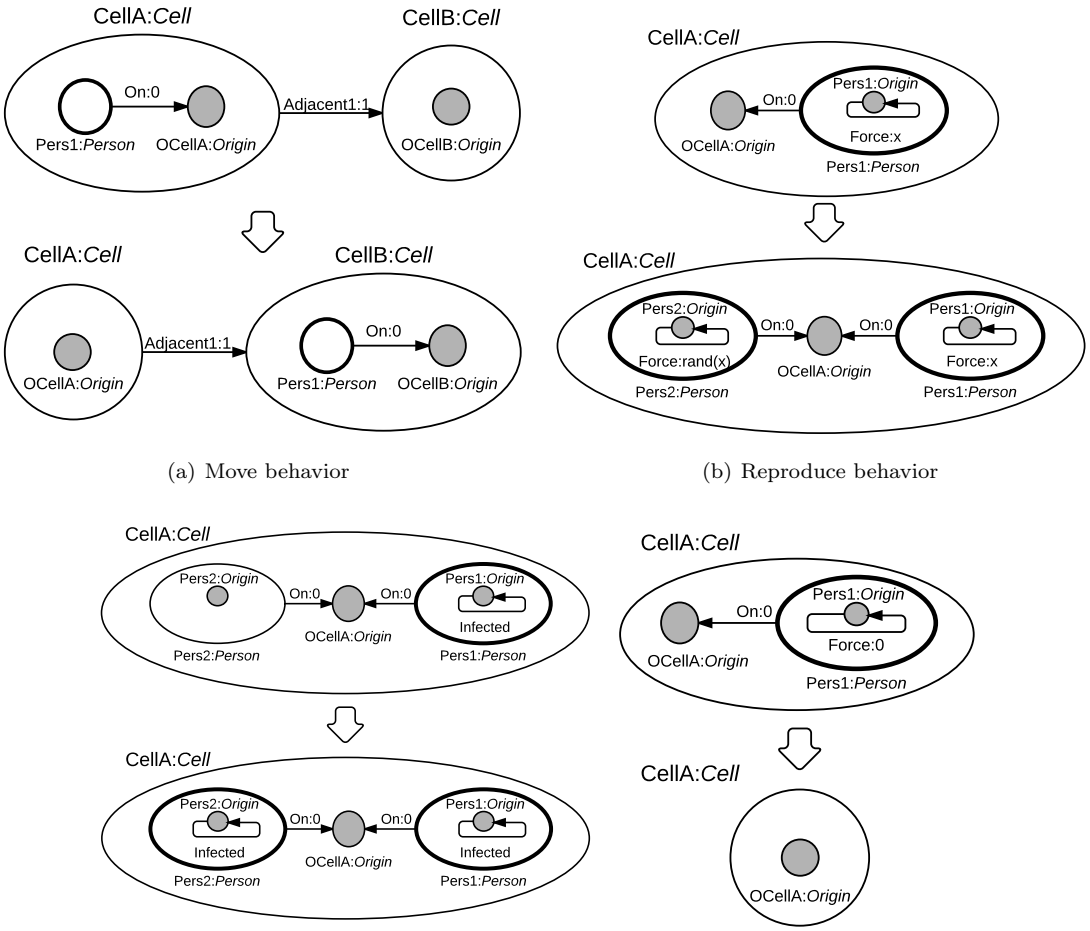
As previously described for the WSP model, Figure 11 shows a workflow which defines the order in which the behaviors of person agents are applied at each time-step.

# 5 Simulation distribution using Nested Graphs

We have presented the use of Nested Graphs to model agents and agent behaviors, we now focus on the other advantage of this method: the distribution of a model on parallel or distributed platforms. In this section we first detail the formal basis of the distribution, then we illustrate the distribution with the WSP model and we compare the method to classical grid modeling.

## 5.1 Formal description

The distribution of the simulation is based on a weight assigned to each agent. Formally, this corresponds to the definition of a weight function $\mathcal{W} : \tau \to \mathbb{N}$ assigning to each agent a weight. This weight function is actually defined by the modeler which assigns a weight to each type of agent. To perform the distribution we compute a density for each vertex with respect to the weights associated with the agents it contains. The density of a Agent Graph $a = \langle \mathrm{G}, \mathrm{A}, \mathrm{T}, \mathrm{L}, \mathrm{V} \rangle$ is defined by the following density function $\mathcal{W}^* : \mathbb{G} \mapsto \mathbb{N}$ such that:




(a) Move behavior

(b) Reproduce behavior

(c) Infectious behavior

(d) Die behavior

Figure 10: Diagram state transition of a person agent behavior in the virus model

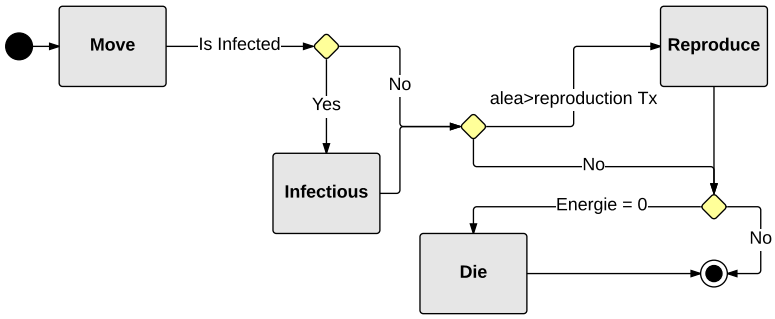



Figure 11: Behavior representation of a person agent behavior in the virus model

$$\mathcal{W}^*(a) = \Sigma_{n \in \mathrm{G}} \mathcal{W}^*(n) + \mathcal{W}(a) \tag{2}$$

This is illustrated in the WSP model where there are three levels of abstraction as shown in Figure 12. The higher level of abstraction represents the parts of the simulation which are executed on each processor. This level of abstraction is necessary to support the distribution of

the model, for this reason, it is considered as the level of abstraction numbered 0. The second level of abstraction is the environment (Cells) on which the simulation is performed. Cells are a way to enfold other agents contained in the simulation but do not implement any behavior hence their weight for the computation distribution is equal to 0. The thrid level is composed of the other agents in the simulation, i.e. Wolfs and Sheep. As they are composed of the same number of behaviors to perform their weight is equal to 1. As the weight of a cell is equal to 0, the density of each cell is equal to the sum of the vertex weights which compose it, the number of wolves and sheep. At the higher level of abstraction the weight values of the cells are used to balance the load between processors as, thanks to their Agent Graph densities, we quantitatively know the amounts of resources needed to perform the computation of a simulation step with respect to Agent Graphs.

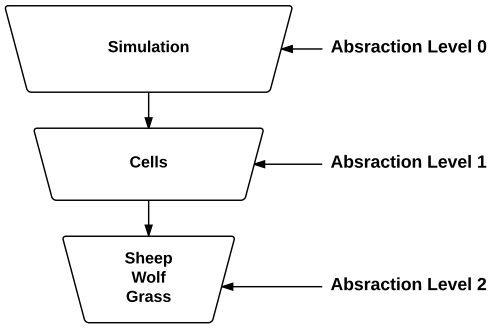



Figure 12: Abstraction levels for the WSP model

Let, $\Omega = \langle \mathrm{G}, \mathrm{A}, \mathrm{T}, \mathrm{L}, \mathrm{V} \rangle$ be the simulation. We define the density graph of $\Omega$ as a weighted graph $\mathcal{D}_\Omega = \langle V, E, W_V, W_E \rangle$ where: $V = \mathrm{G}$ is a set of vertices, $E = \mathrm{A}$ is a set of directed edges, $W_V : V \mapsto \mathbb{N}$ such that $\forall\ a \in \mathrm{G}, W_V(a) = \mathcal{W}^*(a)$ is a function assigning a weight to each node according to the density function $\mathcal{W}^*$, and $W_E : V \mapsto \mathbb{N}$ such that $\forall\ a \in \mathrm{A}, W_E(a) = 1$ is a function assigning a weight to each directed edges. It follows that the distribution of the simulation $\Omega$ among $k$ processors corresponds to the $k$-partitioning of $\mathcal{D}_\Omega$ where the vertices of the $i^{th}$ partitions are mapped to the $i^{th}$ processors.

Indeed, as the $k$-partitioning of a weighted graph is its partitioning into $k$ partitions such that vertice's weights are balanced across partitions, the computational load of the simulation is efficiently distributed among the $k$ processors. Moreover, as the $k$-partitioning of a weighted graph also minimizes the weight sum of cut edges, the communication cost associated with the distribution among $k$ processors is also minimized.

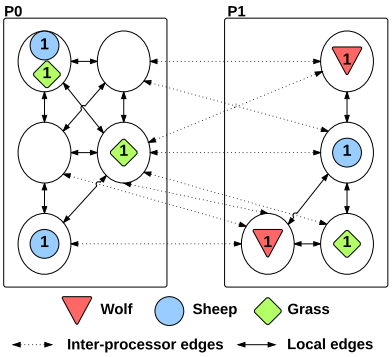
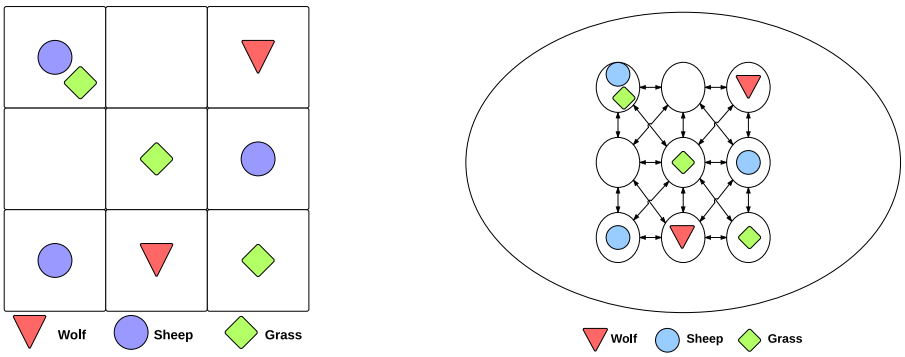



Figure 13: Distribution of the WSP model using Nested Graphs structure on two processors

An arbitrary example of the WSP model distribution where the number of processors $k$ is

equal to 2 is presented in Figure 13.

## 5.2 Distribution illustration




(a) Grid based structure

(b) Nested Graph based structure

Figure 14: Initial configurations of the WSP model

After a formal description of the partitioning problem, we illustrate our method with the WSP model. The classical initial configuration of WSP model environment is based on a grid structure. For instance, an initial configuration, based on $3 \times 3$ grid, is shown on Figure 14(a). It represents the environment on which agents (sheep, wolves) can evolve. Using the Nested Graph method, the corresponding Nested Graph mapping of the initial configuration is given on Figure 14(b). This initial configuration is composed of the three nested levels of graphs corresponding to the levels presented on Figure 12. The first level, the environment, is represented by the main ellipse. It is a container for the two lower levels. The second level represents environment agents in the shape of a grid using vertices and the third level represents wolf and sheep agents.

Using the initial configuration our objective is to run the simulation in parallel. To efficiently use parallel resources we need to divide this initial structure and distribute it on several processors. If we distribute this initial configuration on 2 processors using a Cartesian grid, as it is implemented in most PDMAS platform, we can obtain the two decompositions shown on Figure 15.

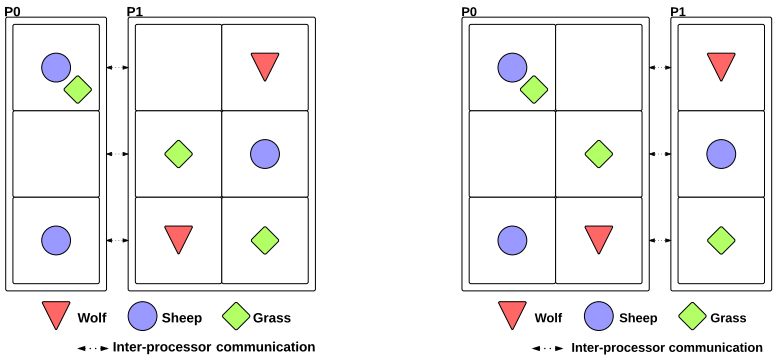



Figure 15: Examples of grid decompositions on the $x$ axis for the WSP model on two processors

As we can see on these figures (Figure 15), the grid based distribution is constrained by the grid structure and the decomposition using the $x$ axis cannot correctly balance the density of agents on each processor. Note that the problem is symmetric with the $y$ axis. As the workload in Multi-Agent Systems is mainly generated by running the agent behaviors, and not by the environment which is usually implemented as data shared between agents, this is of particular importance to

correctly balance the global load. As noted previously this has a direct impact on the performance due to the synchronous execution of time steps.

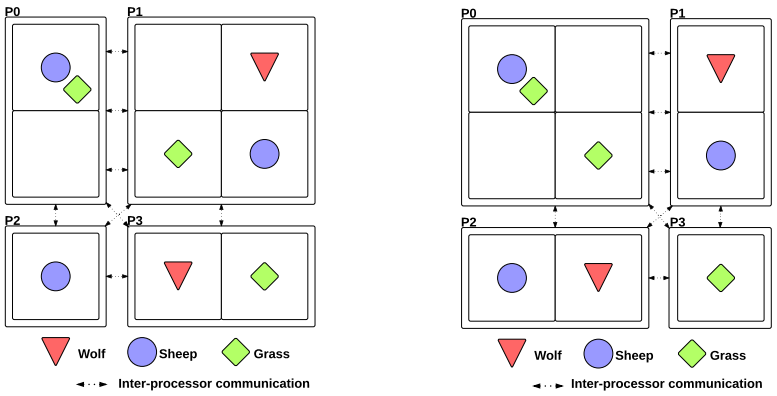


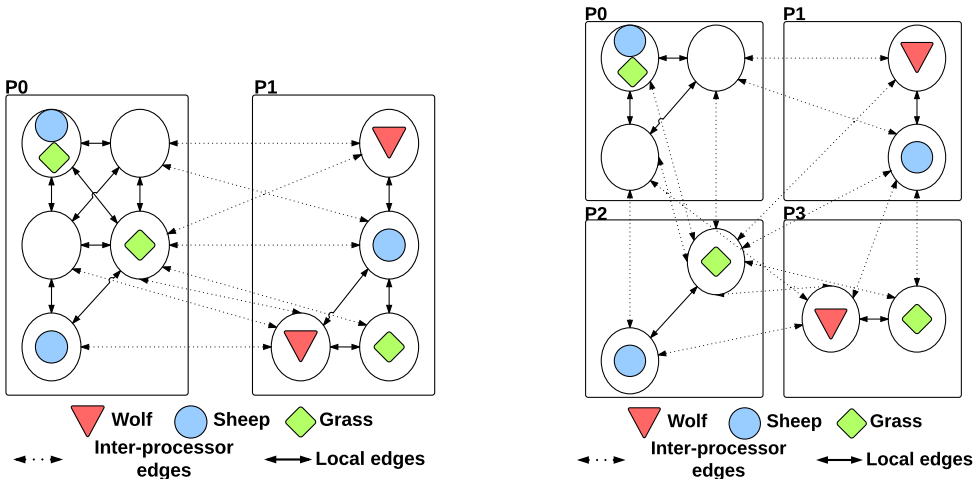

Figure 16: Examples of grid decomposition on $x$ and $y$ axes for the WSP model on four processors

The imbalance is more pronounced if we distribute the initial configuration on 4 processors. Figure 16 represents an example of grid decomposition for the WSP model on 4 processors. Considering the density of agents in these figures, we can note that processors do not have the same workload.

Of course, the presented configurations are too simple to need a parallelization and they are just used to illustrate how a grid-based environment could be distributed using one a axis or two axes decomposition. Using these basic configurations we can, however, more easily imagine the difficulties implied by the distribution of a grid where agents are not uniformly distributed. Obviously, the size of the cell sets mapped to a processor dos not need to be identical on every processor and cut lines may be put where it is necessary to take the agent distribution into account. For instance, on Figures 15 and 16, the decompositions are not uniform in terms of number of cells mapped to each processor. On the other hand, the decompositions are constrained by both the grid cells that cannot be distributed and by the regular structure of the environment as moving an axis to modify the distribution implies to move at least a whole column or row of cells. As a result, the granularity of the decomposition does not allow to achieve a fine balance. It is moreover difficult to dynamically adapt this decomposition when the load varies as the decision of changing the cells mapping may involve several processors.

With the Nested Graph structure, it is easy to compute the distribution based on the density of the agents contained in each vertex of higher level. As there is no rigid structure as a grid behind the environment the distribution is more flexible: every cell can be mapped on whatever processor and the distribution may be introduced at different levels. To illustrate this, Figure 17(a) presents the initial configuration distributed on 2 processors using a Nested Graph structure and a distribution based on density. We can note that the distribution is more balanced between the processors and so more efficient. Figure 17(b) shows the distribution of the WSP model on 4 processors using Nested Graph structure which is also better balanced.

Note that cell distribution and load balance depend on the model that we simulate. In the WSP model the load generated by a sheep is roughly equivalent to the load generated by a wolf so that, reasoning on agents to balance the load, allows to reach a good balance. On the other hand, if the agents generate different loads, we can assign to each agent a weight representing an estimation of the load generated by its execution in order to efficiently balance the distribution. The weights must be assigned by the modeler of the simulation, as an input parameter. Then, to perform the distribution, we compute the density of each cell with respect to the weights associated to the agents it contains, which corresponds to the definition of the weight function.




(a) On two processors

(b) On four processors

Figure 17: Distribution of the WSP model using Nested Graph structure

## 5.3 Distribution tools

One of the advantages of using graph partitioning for the distribution of multi agent simulations based on Agent Graphs is that we can benefit from existing powerful tools. For instance, in our work, we use the Zoltan Framework [4] to perform graph partitioning of the density graphs associated with simulation modeled according to our approach. The Zoltan Framework includes many combinatorial algorithms for parallel scientific applications. This Framework also provides load balancing and dynamic partitioning algorithms that increase parallel application performance by reducing process idle time.

The strength of the Zoltan framework is that it completely separates its inner data structures from the application data structures. This separation is achieved through the use of callback functions (e.g. `ZOLTAN_NUM_OBJ_FN`, `ZOLTAN_EDGE_LIST_MULTI_FN`, etc.). Callback functions are functions written by the user that access its data structures and return the needed data to Zoltan. For example, using callback functions, we can easily get the number of vertices owned by a process or the number of edges for each vertex owned by a process, etc. When an application calls a Zoltan service (e.g. Zoltan_LB_Partition), Zoltan calls these user-provided callback functions to get the application data it needs to realize the partitioning. Integrating the Zoltan framework into an application is done in five main steps: initialization, partitioning, migration, partitioning memory release and memory release as shown in Figure 18.

To validate our proposal we have implemented it in a Parallel and Distributed Multi-Agent platform, called FPMAS for Fractal Parallel Multi-Agent System, that relies on both Agent Graphs and Zoltan. In this platform, every entity is represented by an agent. This platform is only a proof of concept in its current state and is not publicly distributed. We give an overview of its use of Zoltan in Algorithm 1.

In this algorithm, in order to be more efficient, the initial configuration file of the simulation is loaded using an MPI parallel reading. After reading this file in parallel the algorithm calls Zoltan to check the distribution and to update it, if necessary, according to the results returned by Zoltan. Once the initialization is done the algorithm runs the time steps. At each time step the `Zoltan_LB_Balance` function is called to balance the load. This function uses the callback functions to process the global graph and decide which agent has to be migrated.

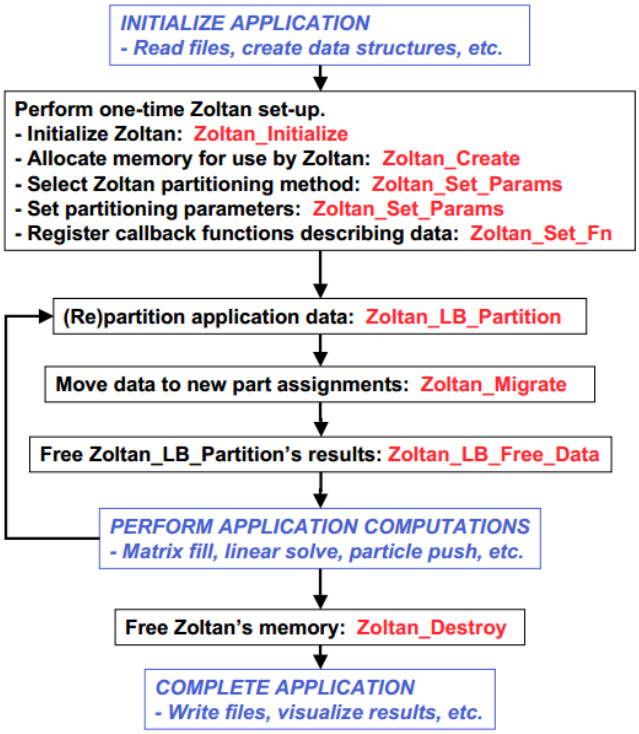



Figure 18: Use of Zoltan into an application [16]).

# 6 Experimentation

In this section, we present results using the Nested Graph structure to model and distribute Multi-Agent simulations. As we want to assess the distribution of multi-agent systems we have used HPC (High Performance Computing) resources for our experimentation. The computing platform provides a large number of cores and allows to validate the scalability of our proposal. It is a 1280 core cluster. Each node of the cluster is a bi-processors, with Xeon E5 (8*2 cores) processors running at 2.6 Ghz frequency and with 4GB of memory per core. The nodes are connected through a non-blocking DDR infiniBand network organized in a fat tree. The computing nodes are shared with other users but the batch system guarantees that the processes are run without sharing their cores.

To assess the performance of the proposal, we have first implemented the WSP model presented in Section 3 on our FPMAS platform. Nevertheless it is not possible to implement the WSP model in all Parallel and Distributed Multi-Agent platforms due to no synchronization support for distant writing calls. For this reason, we have also implemented the reference model defined in [31] to compare the performance of our implementation to other multi-agent platforms

## 6.1 WSP model experimentation

The WSP model used for this experimentation is based on a $1000 \times 1000$ grid environment where 25000 sheep and 17000 wolves are initially randomly positioned. Note that, for reproducibility reasons, the same initial configuration (given in table 1) is used for every simulation, only the seed is changed between two simulations. The model used for the behavior and the energy initialization are taken from the NetLogo implementation [39] of the WSP model. The energy gain for the sheep when eating grass is set to 5 and to 20 for a wolf eating a sheep. The reproduction ratio is set to 5 for the sheep and 4 for the wolves. The simulations are run with 2000 time steps.

Figures 19 and 20 present the execution results of the WSP model based on Nested Graph

**Algorithm 1:** Using Zoltan to manage the distribution of a simulation

```
ReadInitialFileData();
SynchronisationBarrier();
Zoltan_LB_Balance(...);
Zoltan_Migrate(...);
while (!End) do
    Execute_Timestep();
    Zoltan_LB_Balance(...);
    Zoltan_Migrate(...);
end while
```

Table 1: Parameter values for Wolf-Sheep Predation model experimentations

| Parameters | Values set 1 | Values set 2 |
|---|---|---|
| Environment | $1000 \times 1000$ | $1000 \times 1000$ |
| Nb. sheep | 25000 | 25000 |
| Nb. wolf | 17000 | 17000 |
| Sheep reproduction rate | 0.5 | 0.5 |
| Wolf reproduction rate | 0.4 | 0.4 |
| Gain food sheep | 4 | 4 |
| Gain food wolf | 20 | 20 |
| Init. life sheep | 4 | 4 |
| Init. life wolf | 20 | 20 |
| Grown grass | 8 | 30 |
| Timestep | 2000 | 2000 |

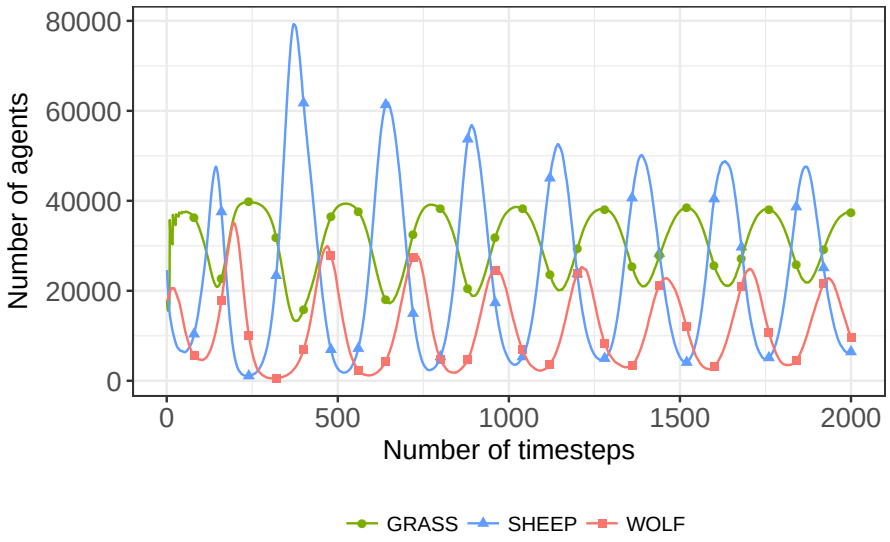



Figure 19: Execution results of the WSP model using Nested Graph structures on 128 cores for 2000 time-steps (Grass grows each 8 time-step)

structures on 64 cores. Figure 19 represents a model with grass growing every 8 time-steps whereas Figure 20 represents a model with grass growing every 30 time-steps. As we can see, the balance of the ecosystems is respected even if the trend is different from a figure to another. This validates that our approach provides the same results as a standard implementation of the model.

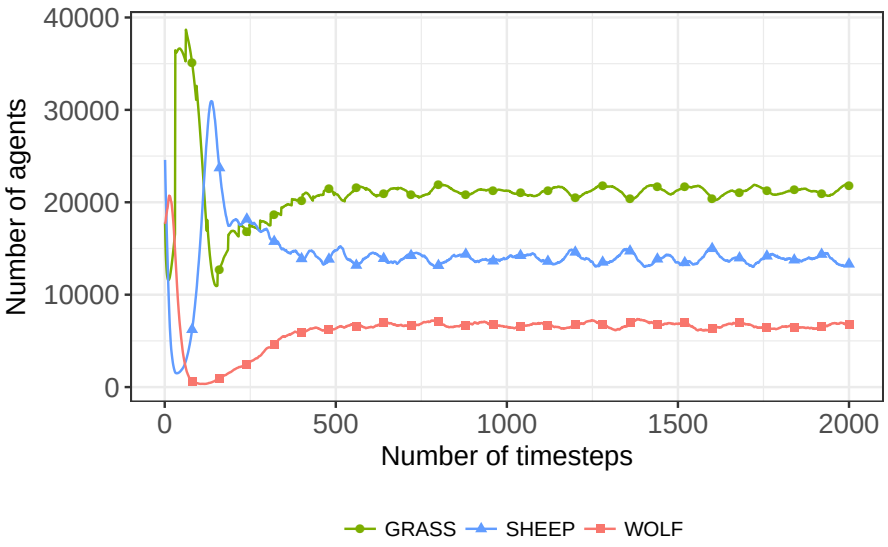
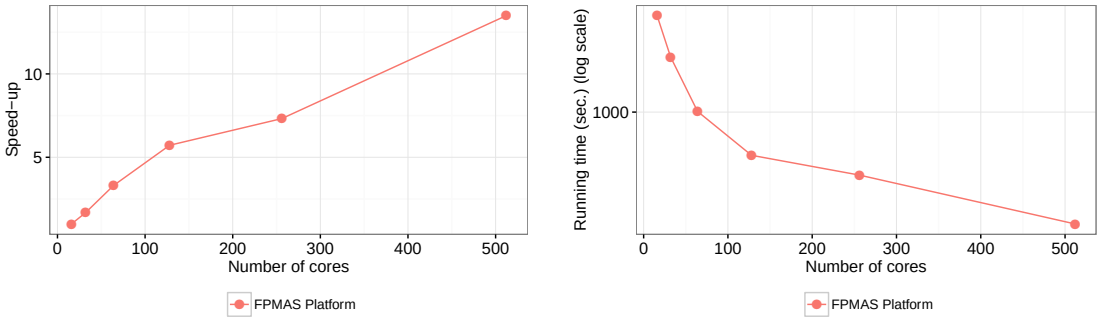



Figure 20: Execution results of the WSP model using Nested Graph structures on 128 cores for 2000 time-steps (Grass grows each 30 time-step)




(a) Speed-up

(b) Running time

Figure 21: Scalability of FPMAS for WSP model using Nested Graph structures from 2 cores to 512 cores for 2000 time-steps

As we target the parallelization of, possibly large, multi-agent systems, scalability is an important property. It guarantees that the approach stays valid even for large models. Figure 21 presents the scalability for the WSP model using the Nested Graph modeling with the FPMAS platform. On the curves every point is the mean value of 10 execution run times. No standard variation is shown in the graphic as the variation ranges are between 0.1 and 0.6. Note that the reference time used to compute the speed-up is based on the 2 core runs and this later cannot thus be more than half the number of cores. This is because FPMAS is a parallel implementation that cannot run on just one core as it includes synchronization. As we can see, the FPMAS platform scales well until 512 cores for the WSP model as there is no break in the speedup progression, although the speedup is not optimal due to a large number of synchronizations.

## 6.2 Virus model experimentation

The Virus model used for this experimentation is based on a $300 \times 300$ grid environment where 9600 persons, including 640 infected, are initially randomly positioned. Note that, like for the WSP model for reproducibility reasons, the same initial configuration (given in table 2) is used for every simulation. Only the seed is changed between two simulations. The model used for the behavior and the energy initialization is taken from the NetLogo implementation [40] of the Virus model. The infection and recovery rates are respectively set to 0.65 and 0.5. The reproduction rate is set to 0.2. The simulations are run with 800 time steps.

Table 2: Parameter values for Virus model experimentations

| Parameters | Value |
|---|---|
| Environment | $300 \times 300$ |
| Env. capacity | 800000 |
| Nb. person. init. | 9600 |
| Nb. person. infec. init | 640 |
| Infection rate | 0.65 |
| Recovery rate | 0.5 |
| Reproduction rate | 0.2 |
| Nb. timestep | 800 |

Figure 22 presents the execution results of the Virus model based on Nested Graph structures on 64 cores with the FPMAS platform. As we can see, the balance between the populations (i.e. Immune and Infected Agents) becomes stable after 420 times steps. This is coherent with the NetLogo results and hence validates the parallel implementation.

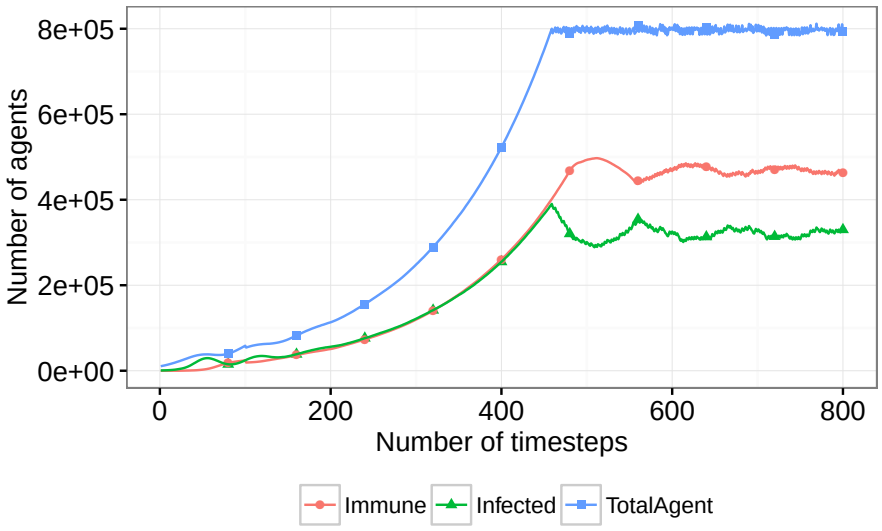
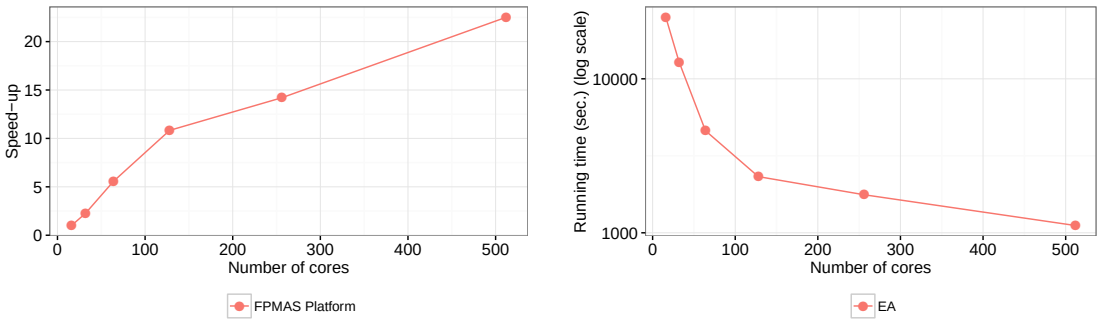



Figure 22: Execution results of Virus model using Nested Graph structures on 64 cores for 800 time-steps

Figure 23 presents the scalability for the Virus model using the Nested Graph modeling with the FPMAS platform. On the graph every point is a mean value of 10 execution run times. As for the WSP model, the FPMAS platform scales well until 512 cores as there is no break in the speedup progression. Remind that the reference time used to compute the speed-up is based on the 2 core runs and thus cannot be more than half the number of cores. Although a slight slowdown after 128 cores, the speedup is better than with the WSP model as the model is less synchronized.




(a) Speed-up

(b) Running time

Figure 23: Scalability of FPMAS for Virus model using Nested Graph structures from 2 cores to 512 cores for 800 time-steps

## 6.3 Reference model experimentation

The objective of the reference model is to implement a multi-agent model that reproduces the main properties usually found in Multi-Agent Systems:

- perception, to interact with the environment and other agents,
- communication, to communicate with other agents or the environment,
- mobility, to move on the environment.

In the reference model, the environment is represented by a square grid. Agents are mobile and move randomly on the grid. A perception field, characterized by the "radius" property, is associated with each agent. It represents the limited perception of the agent on the environment.

Each agent is composed of 3 behaviors:

- with the walk behavior, an agent moves randomly in one of its 8 Moore neighbor cells on the grid (or less if the agent is close to a border). This behavior is used to test the mobility and the perception of the agents,
- with the interact behavior, agents interact and send messages to all the agents in their perception field. This behavior simulates communications between agents and evaluates the communication support of the platforms,
- with the compute behavior, agents compute a "Fast Fourier Transform (FFT)" [18] in order to generate a workload. This behavior simulates the load generated by the execution of the agent inner algorithms.

The global agent behavior consists in performing each of these three behaviors at each time step. The reference model has several parameters that determine the agent behavior and also the global model properties. For instance, the model allows to vary the workload using different sizes of input for the FFT computation. It is also possible to generate more or less communications between agents by setting the number of contacted agents in the interact behavior or to assess the agent mobility by setting the agent speed in the walk behavior.

Figure 24 represents the walk behavior using our modeling method. Note that the interact and compute behaviors are not represented during the modeling phase because they do not perform any transformation on the graph structure. They are only calculus or message exchanges.

To assess scalability we vary the number of cores used to execute the simulations while we fix the number of agents. Results presented on Figure 25 shows the speed-up obtain by three

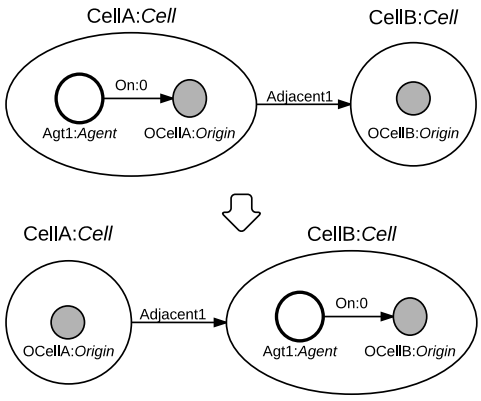



Figure 24: Walk behavior representation of an agent

PDMAS (RepastHPC, Flame and FPMAS) with a 10 000 agent model. The reference model used is based on a $300 \times 300$ grid environment where 10000 agents are initially randomly positioned. Note that the configuration for the RepastHPC platform cannot be initialized with a file and must be done with an initialization function. As this is done in parallel it is not guaranteed that two initializations are strictly the same. For Flame and FPMAS the initializations are based on a file. The perception radius is set to 3 for the mobile agents. All the random laws used (choice of neighbor cell and the initialization of the DFT table) are uniform laws. The size of the DFT table is set to 128. The simulations are run with 200 time steps. In the curves every point is a mean value of ten execution run times. No standard variation is given on the graphic as its ranges between 0.1 and 0.7. Note that the reference time used to compute the speed-up is based on the 2 core runs.

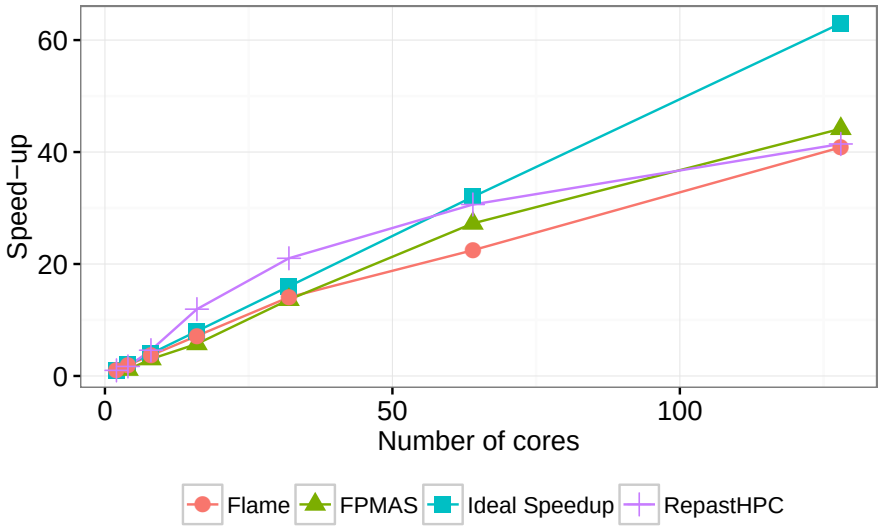



Figure 25: Scalability of FPMAS, RepastHPC, FLAME simulations using 10 000 agents

From Figure 25 we can conclude that the three platforms scale well up to 32 cores but that the RepastHPC speedup tends to increase slower than the other platforms over 64 cores. FPMAS is above the other platforms for more than 100 cores. It even reaches a better speedup than for the WSP model in this case. This validates that our approach, based on the modeling of

MAS with Nested Graphs, scales well. Note that the RepastHPC results are above the ideal speedup for simulations with less than 64 cores. We suspect that these better results come from cache optimizations in the system that favor larger simulations and that the 2 core run has poor performance. As the FPMAS platform is currently in a proof of concept state it could still benefice from optimizations. It is then encouraging to note that it outperforms other platforms when using more than 100 cores. This also means that the Nested Graph approach is suited for parallel implementation of multi-agent simulations.

# 7 Conclusions

In this paper, we propose Nested Graphs as an approach to model Parallel and Distributed Multi-Agent simulations. This method aims at facilitating the dynamic distribution of simulations among parallel machines. This is achieved thank to finer granularity on multiple levels of abstraction. Our contribution is a common and generic framework which represents the agent models as well as their distribution. In addition, this framework includes a more graphical method to model Parallel and Distributed Multi-Agent Simulations.

In our future work, we intend to precisely examine the efficiency of synchronization mechanisms using Nested Graph structures in parallel platforms and propose a platform which includes Nested Graph structure as a modeling and distribution paradigm. In addition, we will develop the formal basis describing our proposal of Nested Graphs applied to Parallel and Distributed Multi-Agent Systems.

# Acknowledgment

This work was supported in part by the ANR DATAZERO (contract ANR-15-CE25-0012) project and by the Labex ACTION project (contract ANR-11-LA BX-01-01).

Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

The C++ implementation of Nested Graphs is available at https://github.com/LoW12/FractalGraph

# References

[1] Elaini S Angelotti, Edson E Scalabrin, and Bráulio C Ávila. Pandora: a multi-agent system using paraconsistent logic. In *Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001.*, pages 352–356. IEEE, 2001.

[2] Matthew Berryman. Review of software platforms for agent based models. Technical report, DTIC Document, 2008.

[3] N Bezirgiannis. Improving performance of simulation software using haskells concurrency & parallelism. Master's thesis, Utrecht University, 2013.

[4] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[5] Rafael H Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J Gomez-Sanz, Joao Leite, Gregory MP O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

[6] Michele Carillo, Gennaro Cordasco, Rosario De Chiara, Francesco Raia, Vittorio Scarano, and Flavio Serrapica. Enhancing the performances of d-mason - a motivating example. In Nuno Pina, Janusz Kacprzyk, and Mohammad S. Obaidat, editors, *SIMULTECH*, pages 137–143. SciTePress, 2012.

[7] Gregory Carslaw. *Agent based modelling in social psychology.* PhD thesis, University of Birmingham, 2013.

[8] Jose R Celaya, Alan Desrochers, Robert J Graves, et al. Modeling and analysis of multi-agent systems using petri nets. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1439–1444. IEEE, 2007.

[9] Cédric Chevalier and François Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *CoRR*, abs/0907.1375, 2009.

[10] Sebastien Chipeaux, Fabrice Bouquet, Christophe Lang, and Nicolas Marilleau. Modelling of complex systems with aml as realized in miro project. *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 3:159–162, 2011.

[11] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn Chin, David Worth, and Chris Greenough. Exploitation of hpc in the flame agent-based simulation framework. In *Proceedings of the 2012 IEEE 14th Int. Conf. on HPC and Communication & 2012 IEEE 9th Int. Conf. on Embedded Software and Systems*, HPCC '12, pages 538–545, Washington, DC, USA, 2012. IEEE Computer Society.

[12] Nicholson Collier and Michael North. *Repast HPC: A platform for large-scale agentbased modeling.* Wiley, 2011.

[13] Nick Collier. Repast hpc manual, 2010.

[14] Gennaro Cordasco, Rosario Chiara, Ada Mancuso, Dario Mazzeo, Vittorio Scarano, and Carmine Spagnuolo. A Framework for Distributing Agent-Based Simulations. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *Lecture Notes in Computer Science*, pages 460–470, 2011.

[15] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science & Engineering*, 4(2):90–96, 2002.

[16] Karen D Devine, Erik G Boman, Lee Ann Riesen, Umit V Catalyurek, and Cédric Chevalier. Getting started with zoltan: A short tutorial. *Sandia National Labs Tech Report SAND2009-0578C*, 2009.

[17] Jacques Ferber and Jean-François Perrot. *Les systèmes multi-agents: vers une intelligence collective.* InterEditions Paris, 1995.

[18] Matteo Frigo and Steven G Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[19] Olivier Gutknecht and Jacques Ferber. Madkit: A generic multi-agent platform. In *Proceedings of the fourth international Conf. on Autonomous agents*, pages 78–79. ACM, 2000.

[20] Lynne Hamill and Nigel Gilbert. Simulating large social networks in agent-based models: A social circle model. *Emergence: Complexity and Organization*, 12:78–94, 2010.

[21] Brian Heath, Raymond Hill, and Frank Ciarallo. A survey of agent-based modeling practices (january 1998 to july 2008). *JASSS*, 12(4):9, 2009.

[22] Bruce Hendrickson and Tamara G Kolda. Graph partitioning models for parallel computing. *Parallel computing*, 26(12):1519–1534, 2000.

[23] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis. *Parallel graph partitioning and sparse matrix ordering library. Version*, 2, 2003.

[24] Mehmet Can Kurt, Sriram Krishnamoorthy, Kunal Agrawal, and Gagan Agrawal. Fault-tolerant dynamic task graph scheduling. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 719–730. IEEE, 2014.

[25] Guillaume Laville, Christophe Lang, Bénédicte Herrmann, Laurent Philippe, Kamel Mazouzi, and Nicolas Marilleau. Mcmas: A toolkit for developing agent-based simulations on many-core architectures. *Multiagent and Grid Systems*, 11(1):15–31, 2015.

[26] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. MASON: A New Multi-Agent Simulation Toolkit. *Simulation*, 81(7):517–527, July 2005.

[27] Sébastien Picault and Philippe Mathieu. An interaction-oriented model for multi-scale simulation. In *IJCAI'2011–Barcelona (Spain)–July, 16-22 2011*, pages 332–337. AAAI Press, 2011.

[28] Alexandra Poulovassilis and Mark Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Transactions on Information Systems (TOIS)*, 12(1):35–68, 1994.

[29] Sivasankaran Rajamanickam and Erik G Boman. An evaluation of the zoltan parallel graph and hypergraph partitioners. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2012.

[30] Vincent Rodin, Abdessalam Benzinou, Anne Guillaud, Pascal Ballet, Fabrice Harrouet, Jacques Tisseau, and Jean Le Bihan. An immune oriented multi-agent system for biological image processing. *Pattern Recognition*, 37(4):631–645, 2004.

[31] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. *A Survey on Parallel and Distributed Multi-Agent Systems*, pages 371–382. Springer International Publishing, 2014.

[32] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. A communication schema for parallel and distributed multi-agent systems based on mpi. In *Euro-Par 2015: Parallel Processing Workshops*, pages 442–453. Springer, 2015.

[33] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review*, 22:27–46, 2016.

[34] Alban Rousset, Bénédicte Herrmann, Christophe Lang, Laurent Philippe, and Hadrien Bride. Using nested graphs to distribute parallel and distributed multi-agent systems. In *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016, Heraklion, Crete, Greece, February 17-19, 2016*, pages 710–717, 2016.

[35] Patrick Taillandier, Duc-An Vo, Edouard Amouroux, and Alexis Drogoul. GAMA: A Simulation Platform That Integrates Geographical Information Data, Agent-Based Modeling and Multi-scale Control. In Nirmit Desai, Alan Liu, and Michael Winikoff, editors, *Principles and Practice of Multi-Agent Systems*, volume 7057 of *Lecture Notes in Computer Science*, pages 242–258. Springer Berlin Heidelberg, 2012.

[36] Seth Tisue and Uri Wilensky. Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of Agent*, volume 2004, pages 7–9, 2004.

[37] Robert Tobias and Carole Hofmann. Evaluation of free java-libraries for social-scientific agent based simulation. *JASS*, 7(1), 2004.

[38] Stéphane Vialle, Eugen Dedu, and Claude Timsit. Parcel-5/parssap: A parallel programming model and library for easy development and fast execution of simulations of situated multi-agent systems. In *Proceedings of SNPD02 International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing*, 2002.

[39] U Wilensky. Netlogo wolf sheep predation model. Center for connected learning and computer-based modeling, Northwestern University, Evanston, IL. Available at: http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation [Accessed April 4, 2017], 1997.

[40] U Wilensky. Netlogo virus model. Center for connected learning and computer-based modeling, Northwestern University, Evanston, IL. Available at: http://ccl.northwestern.edu/netlogo/models/Virus [Accessed April 4, 2017], 1998.