INSTITUT FEMTO-ST

UMR CNRS 6174

Using nested graphs to distribute Parallel and Distributed Multi-Agent Systems

Version 1

Alban ROUSSET — Benedicte HERRMANN — Christophe LANG — Laurent PHILIPPE — Hadrien BRIDE

Rapport de Recherche n° RR–FEMTO-ST–9964

DÉPARTEMENT DISC – February 1, 2016

Using nested graphs to distribute Parallel and Distributed Multi-Agent Systems

Version 1

Alban ROUSSET , Benedicte HERRMANN , Christophe LANG , Laurent PHILIPPE , Hadrien BRIDE

Département DISC

CARTOON

Rapport de Recherche no RR –FEMTO-ST–9964 February 1, 2016 (15 pages)

Abstract: Simulation has become an indispensable tool for researchers to explore systems without having recourse to real experiments. In this context multi-agent systems are often used to model and simulate complex systems. Depending on the characteristics of the modelled system, methods used to represent the system may vary. Whatever the modelling techniques used, increasing the size and the precision of a model increases the amount of computation needed, requiring the use of parallel systems when it becomes too large. Usually, to efficiently run on parallel resources, the model must be adapted to be distributed. In this paper, we propose a new modelling approach, based on nested graphs, that allows the design of large, complex and multi-scale multi-agent models which can be efficiently distributed on parallel resources. A PDMAS (Parallel and Distributed Multi-Agent Platform) that supports this approach and efficiently run parallel multi-agent models is introduced.

Key-words: PDMAS, parallel, distributed platform, Nested Graphs, Modelisation

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tel: (33 3) 81 66 65 15 – Fax: (33 3) 81 66 64 50 – e-mail: fvernill@femto-st.fr

Version 1

Résumé : Pas de résumé

Mots-clés : Pas de mot-clef

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tel: (33 3) 81 66 65 15 – Fax: (33 3) 81 66 64 50 – e-mail: fvernill@femto-st.fr

Using nested graphs to distribute Parallel and Distributed Multi-Agent Systems

February 1, 2016

1 Introduction

Computational simulation is becoming increasingly important even in fields that have not traditionally used computational models such as archaeology, anthropology but also in traditionally fields like psychology [4] or biology [22]. Depending on the characteristics of the modelled system, several methods such as differential equations or Monte-Carlo simulations may be used to represent the system behaviour. 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 behaviour makes it difficult to define a global law modelling the entire system. Based on a simple algorithmic description of individual behaviours, multi-agent systems provides a support to observe emergent behaviours generated by interactions between agents. Recently, the interest for parallel multi-agent platforms has increased. Parallel platforms indeed 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 modelling approach used, increasing the size and the precision of a model increases the amount of computation, requiring the use of parallel systems. While linear systems based models can 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. 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. This is due to the less regular interactions and the more dynamic behaviours of agents. So we can raise a 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. In this paper we assess the use of Nested Graphs (NG) [20] as a data structure to represent agent models. We also show that this data structure provide a simple and efficient way to distribute and parallelize the simulation at different level.

This article is organised as follows. In section 2, we detail some basic knowledge about Multi-Agents Systems and the Parallel and Distributed Multi-Agent platforms (PDMAS) context before identifying the limits regarding distribution of the existing platforms. In section 3, we explain the advantages of using NG structures and we illustrate it on a well-known model. We present some execution results of our proposition in section 6. Finally we present conclusion and future work.

2 Context and related work

A multi-agent system is a platform which provides support to run simulations based on several autonomous agents. Among the most known platforms we can cite NetLogo [24], MadKit [14], Mason [18] and Gama [23]. For large models, these platforms are sometimes no longer sufficient to run the model in terms of memory and computation power. This is, for example, the case if we want to simulate individual behaviour of urban mobility [6] in a large city. Increasing the size or the precision of models could however bring emergent behaviours that we never expected or never seen otherwise. Using parallel systems is a way to exceed these limits in terms of computation power and memory. Several Parallel and Distributed Multi-Agent Platforms (PDMAS) exist, we can cite RepastHPC [8], D-Mason [10], Pandora [2] and Flame [7]. All these platforms provide a native support for parallel execution of Multi-Agent models but also most important mechanisms of distribution, migration and load balancing for the simulation run. In [1] we survey existing PDMAS and compare them with a qualitative analysis and a performance evaluation. Several key points must be enforced for an efficient parallel execution: load balancing, agent migration, communication between agents and coherency in agent vision to cite some of them.

The load, between the processors which participate to the execution, needs to be as much balanced as possible in order to minimize the waiting time between the processors. Indeed, in parallel multi-agent simulations the set of agents is distributed among the processors and each processor runs at its own speed. Multi-agent simulations are synchronous simulations that are driven by time steps. Running one time step may be more or less longer depending on the number of agents mapped to this processor and of the speed of the processor. To keep the consistency of the simulation or computation, all the processors must run in the same time step and synchronization barriers are thus necessary. At each synchronization barrier, at the end of the time step, each processor waits until this synchronization is performed so that the simulation step is bounded by the slowest processor participating to the simulation. To be more efficient and get more performance in terms of running time, the load of the processors must then be as similar as possible all along the simulation.

In an urban mobility model, agents are situated and mobile: they have a position in the environment and they can move on the environment. If the environment is statically 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 agents migrate, the system must follow the agent places in order to deliver messages.

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. Parts of its perception field are recorded in the memory of different nodes. This could be managed, by hand, in the model implementation but this leads to complex algorithmic development. For this reason PDMAS usually tackle this issue by providing parallelized structures. These structures usually provide overlapping areas, areas on the distribution borders are replicated on the neighbour nodes, to avoid to much communication 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 axe x or on two axes x and y such as on the Figures 2, 3. The problem is, even with a fine grain division of the grid as it is

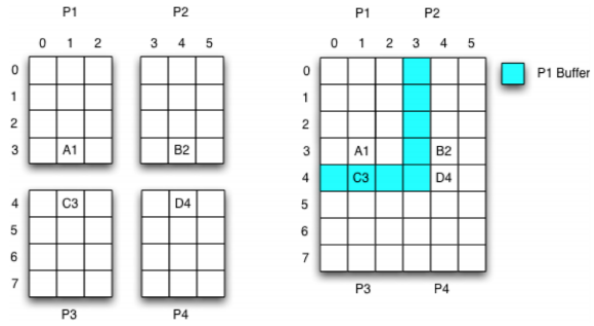

Figure 1: An example of grid decomposition for the RepastHPC platform on four processors with overlapping zone of size 1 [9]

done in the D-MASON platform [3], we always have a square or rectangle distribution which could not be appropriate for all types of models. Moreover this structure may not be flexible enough to correctly balance the load.

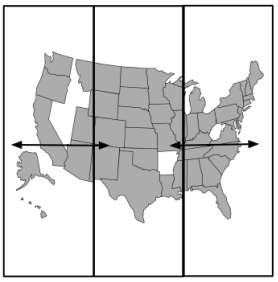

Figure 2: An example of grid decomposition on x axe for the platform D-Mason on three processors [9]

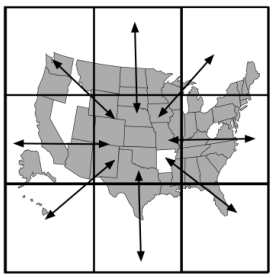

Figure 3: An example of grid decomposition on x and y axes for the platform D-Mason on nine processors [9]

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 [17] or partitioning problem [15]. It exists a lot of tools to partition graphs on parallel or distributed computers such as Parmetis [16] or Zoltan [11]. These tools have great performance results even for large graphs [21]. Taking advantage of these tools is thus possible to improve PDMAS runs on parallel computers. The price to pay is to change the way agent models are represented.

A multi-agent simulation could 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 [23]. Multi-agent models could thus be modelled using graphs with vertices representing agents and edges representing interactions between agents. The modelling job could however be a hard task because the graph structure is very likely to be complex and modelling 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.

We then propose to use a suited structure, based on Nested Graphs, to model a Multi-Agent simulation. This structure natively integrates a more easy-to-use and flexible graph structuring and enhance the support for distribution and load balancing on parallel platforms.

3 Proposition

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.

Nested Graphs are graphs which nodes can be Nested Graphs. They are then recursive structures. The new aspect 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 [20], 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 [19], authors use Nested Graphs to model simulation of complex systems. Using a hierarchical structure is a way to conceptualise more easily the complex system that is to be modelled. We show in the following that Nested Graphs allow the description of any multi-agents system at different levels of abstraction and are natively adapted to be efficiently distributed in a parallel environment. We base our proposition on the two Formal principles for a multi-scale simulation explained in [19]:

- 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 proposition: "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 proposition, we set that all the components participating to the simulation are agents as in [23]. It means that the environment is modelled by one or more agents, depending on the type of environment, similarly 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, set that each agent in the simulation is a typed and labellised Nested Graph, called Agent Graph (Multi-agent Representation Graph). Agent behaviours 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 could for instance be, communication between agents or an agent position relative an environment cell.

Formally, let Γ be a set of types, Σ be a set of labels and Λ a set of values, the set $\mathbb{G}$ of Agent Graphs is recursively defined by $\mathcal{G} \in \mathbb{G} \Leftrightarrow \mathcal{G} = \langle \mathfrak{G}, \mathfrak{A}, \mathfrak{T}, \mathfrak{L}, \mathfrak{V} \rangle$ where $\mathfrak{G} \subseteq \mathbb{G}$ is a set of Agent Graphs (also called nodes) representing the agents of the model, $\mathfrak{A} \subseteq \mathfrak{G} \times \mathfrak{G}$ is a set of directed arcs (or edges) representing relations between agents, $\mathfrak{T} : \mathfrak{G} \mapsto \Gamma$ is a typing function assigning a type to each agent, $\mathfrak{L} : \mathfrak{G} \mapsto \Sigma$ is a labelling function assigning a label to each agent, and $\mathfrak{V} : \mathfrak{A} \mapsto \Lambda$ is a valuing function assigning a value to any arc.

The state of a simulation is fully described by its Agent Graph. Agents behaviours are described as Agent Graph transformations which are modelled using a pair of Agent Graphs with no labelling function and a special Agent graph. Agent Graph transformations can be applied to an agent of the same type as its special place. When apply, if the first Agent Graph can be found as a sub Agent Graph of the Agent Graph of the simulation (i.e. there is a one-to-one correspondence between the first Agent Graph and part of the simulation's Agents Graph), then the found sub Agent Graph is transformed into the second Agent Graph. The first 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 other word we can see a Graph Agent transformation as a conditional structure (*if* [we have this configuration] *then* [we need to arrived to this configuration] *end*) so that the expected behaviour can be performed. When a type of agent have multiple behaviours, its Agent Graph transformations are applied according to a defined work-flow.

Based on these definitions, our proposition is based on two main points:

- A modelling method where multi-agent systems are modelled 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, a special kind of Nested Graphs. Agent behaviours are implemented based on Agent Graph transformations.
- An adapted PDMAS using Agent Graphs models is used to efficiently run the simulation in parallel environment (see Section 5).

With this proposition, complex systems and complex behaviours can be modelled in a graphical way while providing a fully formal basis similarly to Petri Net [5]. It also encompasses multi-level of abstractions needed by the modellers. The proposition is illustrated in the following section with the modelling of a well-known model.

4 Proposition illustration

In this section, we illustrate our proposition with the Wolf-Sheep Predation Model [25] which is a classical model in Multi-Agent Systems.

Let us first introduce some graphical notations: Agent Graphs nodes are represented by ellipses with labels of the form *Label:Type* giving the labels and types of the associated nodes. Edges between nodes are represented by arrows with labels of the form *Type:Value* giving the types and values of the associated edge. An Agent Graph transformation is described by two Agent Graphs linked by a large arrow. We denote by a bold ellipse the special node of an Agent Graph transformation.

The environment of the Wolf-Sheep Predation Model (WSPM) is a grid composed of cells. It represents the first and highest level of abstraction of our simulation. It is modelled using an Agent Graph in which nodes are Agent Graphs of type *Cell* and adjacent nodes are connected by edges of type *Adjacent* (see Figure 4). Any Agent Graph of type *Cell* has a node of type *Origin* which is used to connect agents contained in the same cell.

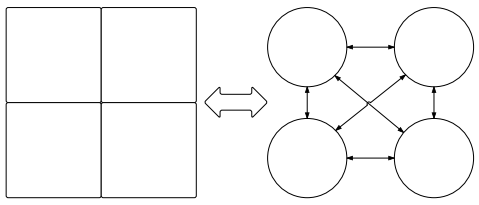

Figure 4: Graph representation of a grid

Cells contain grass, sheeps 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* node 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 node of type *Origin* as well as a real valued arc (from

Origin to *Origin*) of type *Force* which represents the vital force of the considered agent. Agent Graphs of type *Grass* have no characteristics and are therefore empty.

At each timestep of the simulation, the following behaviours modelled as Agent Graph transformations are applied:

- Grass growth (Figure 5): according to a given probability grass appears on a cell.

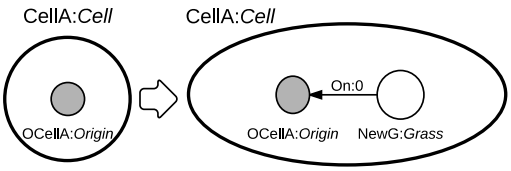

Figure 5: Grown Grass behaviour representation

- Move behaviour (Figure 6): sheeps and wolves randomly move to an adjacent cell.

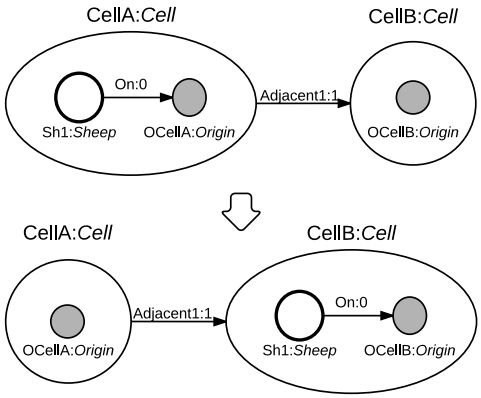

Figure 6: Move behaviour representation of a sheep agent

- Eat behaviour (Figure 7): sheeps (resp. wolves) eat grass (resp. sheeps) and increase their vital force. The grass (resp. sheeps) eaten must be removed from the simulation.

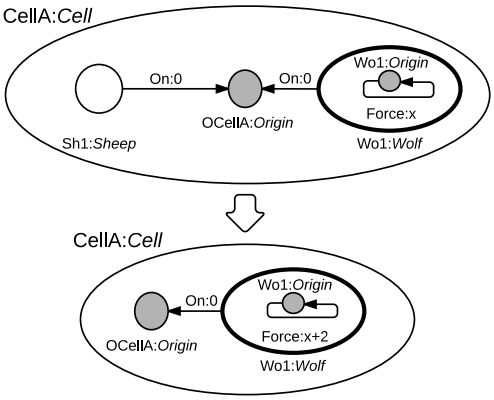

Figure 7: Eat behaviour representation of a wolf agent

- Reproduce behaviour (Figure 8): according to a given probability, sheeps (resp. wolves) reproduce and create new sheeps (resp. wolves) with a default vital force, their vital force is then divided by two.

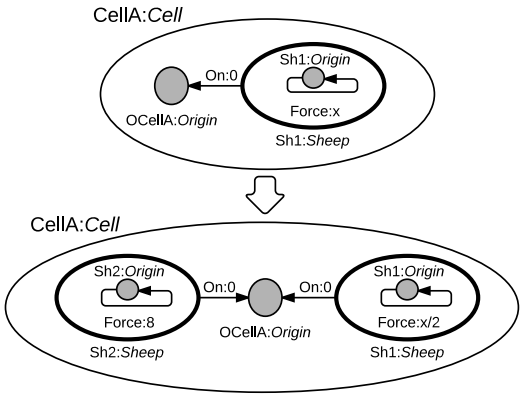

Figure 8: Reproduce behaviour representation of a sheep agent

- Die behaviour (Figure 9): sheeps (resp. wolves) die whenever their vital force drop to zero. They must then vanish from the simulation.

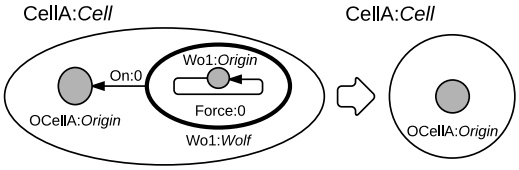

Figure 9: Die behaviour representation of a wolf agent

Figure 10 shows a workflow defining the order in which the behaviours of sheep agents are applied at each timestep. This workflow is almost the same for wolf agents except for the eat behaviour: wolf agents does not eat grass but sheeps.

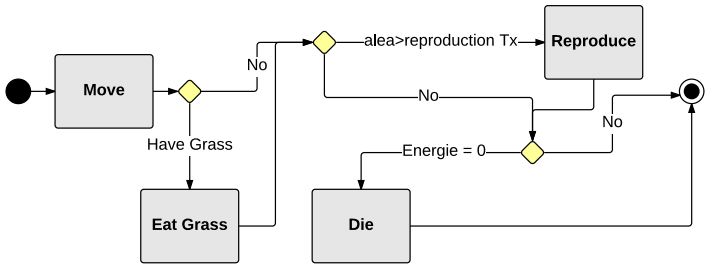

Figure 10: Diagram state transition of a sheep agent behaviour

5 Simulation distribution using Nested Graphs

We have presented the use of nested graphs to model agents and agents behaviours, we then focus on the other advantage of this proposition: the distribution of model in parallel or distributed platform. To illustrate this, we use the same WSPM and we compare our proposition to classical grid modelling. The presented illustration is based on a $3x3$ grid which represents the environment on which agents (sheeps, wolves) can evolve.

Figure 11 represents an arbitrary initial configuration of a WSPM using a grid structure.

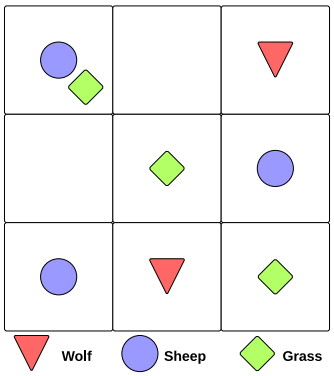

Figure 11: An initial configuration of the WSPM using a grid structure

Using our proposition, the representation of the initial configuration shown in Figure 11 is given in Figure 12. This initial configuration is composed of three nested levels of graphs. The first level, the environment, is represented by the main ellipse and is a container for the lower levels. The second level represents environment agents in the shape of a grid using vertices and the third level represents wolfs and sheep agents.

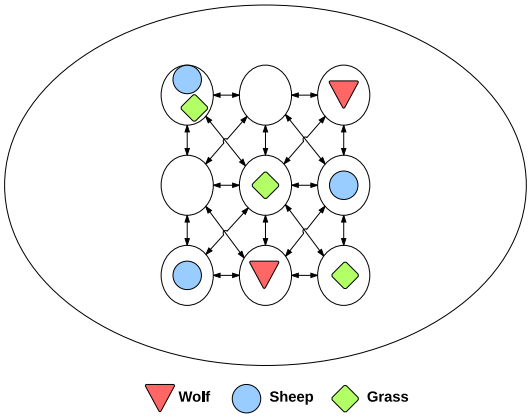

Figure 12: An initial configuration of the WSPM using Nested Graphs structure

Figures 11 and 12 represent the initial configuration to run the model. As we are in a parallel context we need to divide these initial structures to distribute them 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 obtain Figures 13 and 14.

As we can see on these figures, the grid based distribution is not efficient because the density of agents on each processor is not correctly balanced. The workload in Multi-Agent Systems is indeed mainly generated by running the agent behaviours and not by the environment which is actually data. Note that, due to the time step mode of running multi-agents simulations, all the nodes must run synchronously. This means that the slower nodes will determine the running speed of the whole simulation. This is of particular importance to correctly balance the load generated by the simulation as it has a direct impact on the performance.

With the Nested Graphs structure it is easy to compute the distribution based on the density of agents contained in each vertex of higher level. In order to efficiently balance the distribution, we can assign to each agent a weight representing an estimation of the load generated by this agent.

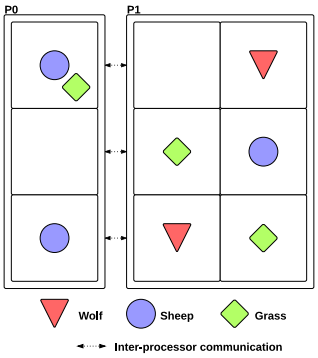

Figure 13: An example of grid decomposition on x axe for the WSPM on two processors

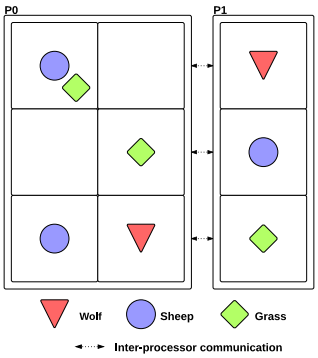

Figure 14: An other example of grid decomposition on x axe for the WSPM on two processors

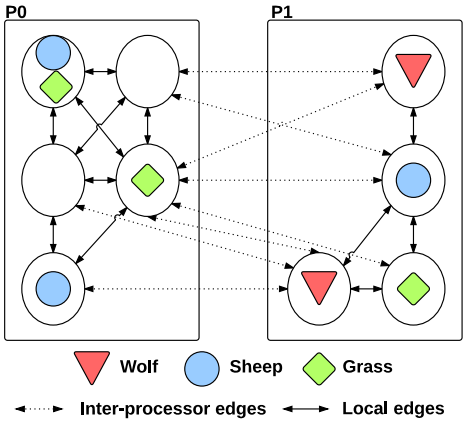

Figure 15: A configuration of the WSPM using Nested Graphs structure distributed on 2 processors

Figure 15 represents the initial configuration on 2 processors using a Nested Graphs structure and a distribution based on density. We can note that the distribution more balanced between processors and so more efficient. Even if it depends on the model that we simulate. The imbalance is more pronounced if we distribute on 4 processors.

Figures 16 and 17 represent an example of grid decomposition for the Wolf-Sheep Predation Model on 4 processors. Considering the density of agents in these figures, we can note that processors do not have the same workload. Whereas a configuration of Wolf-Sheep Predator using Nested Graph structure distributed on 4 processors, as shown on Figure 18, the workload is balanced between the processors.

To perform the distribution we compute the density of each cell in terms of agents using this formula:

$$\sum_{i=0}^{n}\sum_{j=0}^{m} Agtweight_{ij}$$

where i represents the number of hierarchical levels and j represents the number of elements.

One of the interests of using graph based structures is that we can benefit from existing powerfull tools. We use the Zoltan Framework to perform the distribution of the Nested Graph. The strength of the Zoltan Framework is that Zoltan completely separates Zoltan's data structures from application's data structures. This separation is achieved through the use of callback

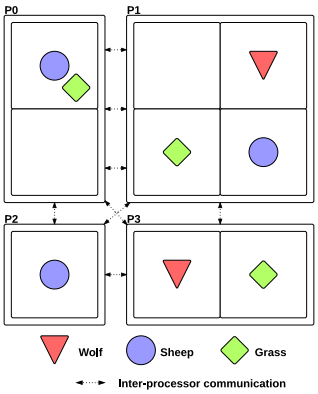

Figure 16: An example of grid decomposition on x and y axes for the WSPM on four processors

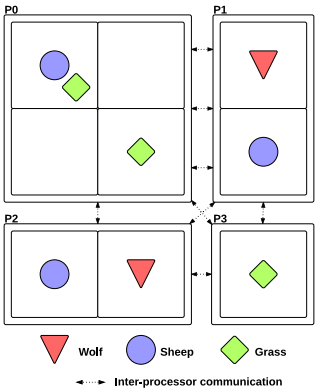

Figure 17: An example of grid decomposition on x and y axes for the WSPM on four processors

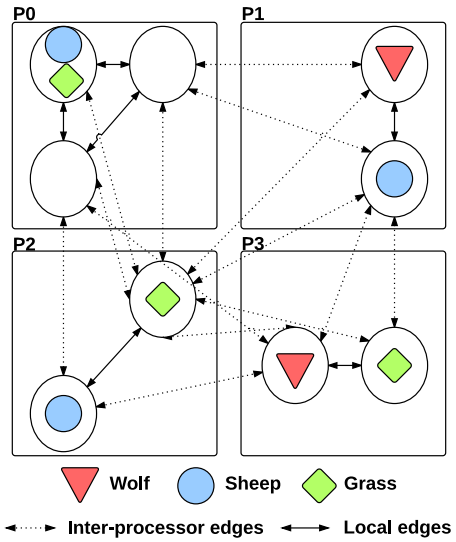

Figure 18: A configuration of the WSPM using Nested Graphs structure distributed on 4 processors

functions (eg. ZOLTAN_NUM_OBJ_FN, ZOLTAN_EDGE_LIST_MULTI_FN, etc.). Callback functions are functions written by the user that access the user's data structures and return 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. Figure 19 represents how a Zoltan Partitioner can be integrated in an application to perform Distribution or Load Balancing.

To assess our proposition, we have developed an implementation of our solution. In the next section we present performance results of two models with our implementation.

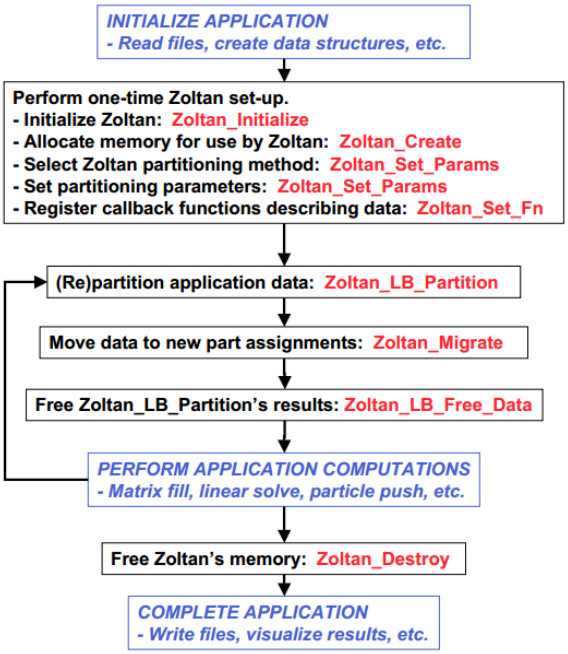

Figure 19: Use of Zoltan in a typical dynamic application. Calls to Zoltan functions are prefixed by Zoltan_ [12]

6 Experimentation

In this section we present some results using the Nested Graph structure to model and distribute Multi-Agent simulation. We implemented our proposition in our Parallel and Distributed Multi-Agent Platform. To assess the performance of our proposition, we have implemented the WSPM presented in Section 3.

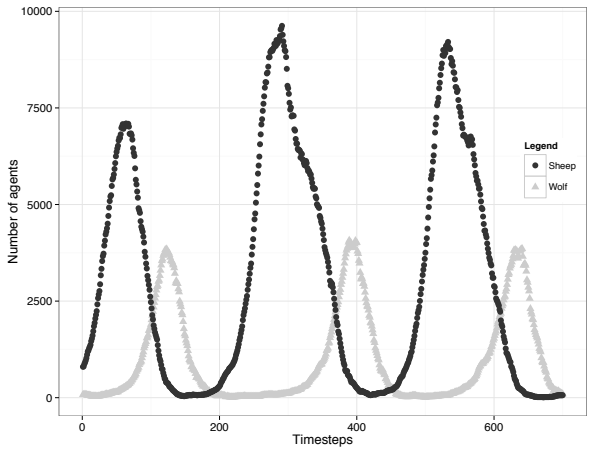

Figure 20: Execution results of the WSPM using Nested Graphs structures on 8 cores (Initial configuration: 800 sheeps and 80 wolfs

Figure 20 presents the execution results of the WSPM using Nested Graphs structures on 8 cores. As we can see, the balance of the ecosystems is respected.

Nevertheless as the WSPM is not implemented in other Parallel and Distributed platform and to compare our work to other platforms we decided to implement a reference model defined in [1] to exhibit the performance of our solution. This reference model respects the main properties that can be found in Multi-Agent Systems: (i) perception, to interact with the environment and other agents, (ii)communication, to communicate with other agents or environment, and (iii) mobility, to move on the environment.

In this model the environment is represented by a square grid. Agents are mobile and move randomly on the grid. A perception field, characterised 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 behaviours: (i) with the walk behaviour agents move in a random direction on the environment. This behaviour is used to test the mobility and the perception of the agents, (ii) with the interact behaviour agents interact and send messages to other agents in their perception fields. This behaviour simulates communications between agents and evaluates the communication support of the platforms, (iii) with compute behaviour agents compute a "Fast Fourier Transform (FFT)" [13] in order to generate a workload. This behaviour simulates the load generated by the execution of the agent inner algorithms.

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

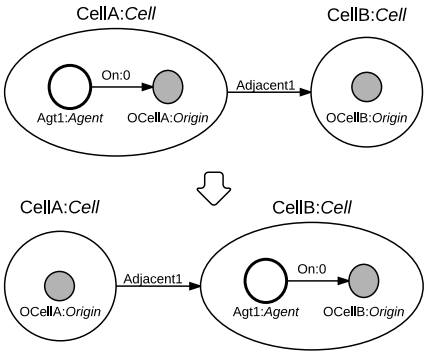

Figure 21: Walk behaviour representation of an agent

Figure 21 represents the walk behaviour using our modelling proposition. The interact and compute behaviours are not represented during the modelling phase because they does not perform any transformation on the graph structure. They are only calculus or messages sent/received.

About the HPC experimental settings, we have run the reference model on a 764 cores 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 32 Go of memory. The nodes are connected through a non blocking DDR infinyBand network organised in a fat tree. The system is shared with other users but the batch system guaranties that the processes are run without sharing their cores.

We have realised several executions in order to exhibit the MAS behaviours concerning scalability (Figures 22). To assess scalability we vary the number of nodes used to execute the simulations while we fix the number of agents. We then compute the obtained speed-up. Each execution is realised several times to assess the standard variation and the presented results are

the mean of the different execution durations. Due to a low variation in the simulation runtime, the number of executions for a result is set to 10.

Execution results on the scalability of a 10 000 agents model are given on Figure 22. Note that the reference time used to compute the speed-up is based on a two cores run.

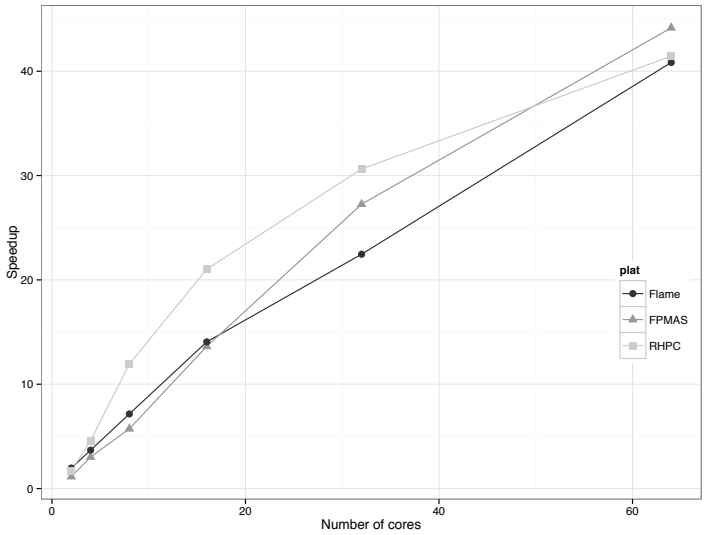

Figure 22: Scalability of FractalPMAS, RepastHPC, FLAME simulations using 10 000 agents, FFT 100 and 200 cycles

This is due to RepastHPC which cannot run on just one core so that its reference time must be based on two core runs. The speedup is therefore limited to half the number of nodes. We can note that the three platforms scale well up to 32 cores In addition on Figure 22 we can see that RepastHPC results are above the ideal speedup for simulations with less than 50 cores. As we suspected that these better results come from cache optimizations in the system, we did more tests to confirm this hypothesis. On the Figure 22, FPMAS is above other platforms for more than 55 cores.

The Figure 22 shows that our proposition with NG scale well and can be compared to other platforms. It's important to notify that FractalPMAS is a proof of concept and that a lot of optimization should be done.

7 Conclusion

In this paper, we present a new approach to model Multi-Agents Systems based on Nested Graphs as a data structure to model Parallel and Distributed Multi-Agent simulation. This structure allows us to dynamically distribute the simulations amoung parallel machines with finner granularity on multiple levels of abstraction. Our contribution aims at proposing a common and generic framework which represent the agent models as well as their distribution. In addition, this framework define a more graphical method to model Parallel and Distributed Multi-Agent Simulations.

In our future work, we intend to precisely examine the efficiency of synchronisation mechanisms using Nested Graphs structure in parallel platforms and propose a platform which includes Nested Graphs structure as a modelling and distribution tool. In addition, we are preparing a formal basis describing our proposition of Nested Graphs applied to Parallel and Distributed Multi-Agent Systems.

Acknowledgment

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

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

References

[1] Alban Rousset Bénédicte Herrmann Christophe Lang Laurent Philippe. A survey on parallel and distributed multi-agent systems. 8805, 8806, 08 2014.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] U Wilensky. Netlogo wolf sheep predation (docked) model. *Online] Center for connected learning and computer-based modeling, Northwestern University, Evanston, IL. Available at: http://ccl. northwestern. edu/netlogo/models/WolfSheepPredation (docked).[Accessed 2 June 2009]*, 2005.

FEMTO-ST INSTITUTE, headquarters
15B Avenue des Montboucons - F-25030 Besançon Cedex France
Tel: (33 3) 63 08 24 00 – e-mail: contact@femto-st.fr

FEMTO-ST — AS2M: TEMIS, 24 rue Alain Savary, F-25000 Besançon France
FEMTO-ST — DISC: UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST — ENERGIE: Parc Technologique, 2 Av. Jean Moulin, Rue des entrepreneurs, F-90000 Belfort France
FEMTO-ST — MEC'APPLI: 24, chemin de l'épitaphe - F-25000 Besançon France
FEMTO-ST — MN2S: 15B Avenue des Montboucons - F-25030 Besançon cedex France
FEMTO-ST — OPTIQUE: 15B Avenue des Montboucons - F-25030 Besançon cedex France
FEMTO-ST — TEMPS-FREQUENCE: 26, Chemin de l'Epitaphe - F-25030 Besançon cedex France

http://www.femto-st.fr