Overview of Case Studies on Adapting MABS Models to GPU Programming

Emmanuel Hermellin(✉) and Fabien Michel

LIRMM - CNRS University of Montpellier,
161 Rue Ada, 34095 Montpellier, France
{hermellin,fmichel}@lirmm.fr

Abstract. General-Purpose Computing on Graphics Units (GPGPU) is today recognized as a practical and efficient way of accelerating software procedures that require a lot of computing resources. However, using this technology in the context of Multi-Agent Based Simulation (MABS) appears to be difficult because GPGPU relies on a very specific programming approach for which MABS models are not naturally adapted. This paper discusses practical results from several works we have done on adapting and developing different MABS models using GPU programming. Especially, studying how GPGPU could be used in the scope of MABS, our main motivation is not only to speed up MABS but also to provide the MABS community with a general approach to GPU programming, which could be used on a wide variety of agent-based models. So, this paper first summarizes all the use cases that we have considered so far and then focuses on identifying which parts of the development process could be generalized.

Keywords: MABS · GPGPU · GPU delegation

1 Introduction

Multi-Agent Based Simulation (MABS) rely on directly modeling and simulating the interactions of micro-level entities, namely agents [15]. In this scope, when a large number of agents have to be modeled, the simulation process may require a lot of computing resources, which often represents a major issue with respect to the experimentation [17].

Considering this issue, General-Purpose Computing on Graphics Units (GPGPU) is a High Performance Computing (HPC) solution which is interesting because it allows to use the massively parallel architecture of the graphics cards of usual computers for accelerating general-purpose computations in a very efficient way[1] [5,6]. However, GPGPU is strongly related with the highly specialized architecture of Graphics Processing Units (GPU) and thus implies a specific programming approach to be used efficiently. In particular, a MABS

[1] e.g. https://developer.nvidia.com/about-cuda.

J. Bajo et al. (Eds.): PAAMS 2016 Workshops, CCIS 616, pp. 125–136, 2016.
DOI: 10.1007/978-3-319-39387-2_11

model cannot take advantage of the GPU power without being adapted to the GPU programming paradigm.

Still, it turns out that adapting a MABS model to GPU programming is a difficult task which is generally done only for the sake of efficiency, in an adhoc way, and especially without considering the accessibility nor the reusability of the produced software [20]. Indeed, to be GPU compliant, the problem must be represented by distributed and independent data structures and usual object oriented features, which are common in MABS, are no longer available using GPGPU [16]. Therefore, most of the works that use GPGPU for developing MABS are hardly accessible and thus not reusable outside of the context for which they have been designed. So, a lot of development efforts are lost and the use of GPGPU still represents a marginal approach in the scope of MABS despite its increasing success as an HPC solution.

Among the research works that tackle the use of GPGPU for MABS from a software engineering perspective, most of them propose solutions that hide GPU programming, thus making its use completely transparent (e.g. [12,25]). However, because of the heterogeneity of multi-agent models, such solutions only focus on some particular type of MAS models and cannot be easily generalized.

Instead of tackling accessibility and reusability by means of transparency, [14] proposes a design guideline which promotes and eases the direct use of GPU programming when adapting and developing MABS models in a GPGPU context, namely the GPU environmental delegation principle. More specifically, this approach relies on identifying which parts of the MAS model could be implemented using GPU programming. So far, we have used this approach on six different use cases [9,10,14] and obtained interesting results both from a performance point of view and from a software engineering perspective. This paper first summarizes these different works and their results before discussing the practical lessons learned from these experiments. Especially, based on the idea that there is a need for a methodology for adapting MABS to GPU programming, we show that we have been able to identify a recurrent development pattern which could be generalized.

The paper is organized as follows. Section 2 presents GPGPU basics and the evolution of the use of GPGPU in MABS. Section 3 describes and summarizes all the experiments done with the GPU delegation principle. Section 4 presents a first generalization of the approach toward the creation of a methodology dedicated to GPGPU for MABS. Section 5 concludes the paper and outlines some related perspectives.

2 Related Works and Motivations

2.1 GPGPU-Based MABS

Initially designed for graphics rendering, GPU are now composed of hundreds of ALU (Arithmetic Logic Units) forming a highly parallel structure able to perform general-purpose computations. The programming paradigm associated consists in executing simultaneously a series of operations on a dataset. When the data structure is suitable (and only if), the massively parallel architecture of the GPU provides very high performance gains (thousands of times faster).

In the context of MABS, empirical results from various experiments show that high simulation speeds can be achieved with very large agents populations (e.g. [7]). Nonetheless, this excellent speedup comes at the expense of modularity, ease of programmability and reusability [20]. This was especially true at the beginning of GPGPU because there was no specialized programming interface and one had to hack the GPU graphical functions to achieve GPGPU-based simulations [23].

Still, despite the evolution of GPGPU, and especially the release of specialized programming frameworks such as CUDA[2] and OpenCL[3], the current GPGPU-based MABS solutions remain difficult to use and it is worth noting that the majority of the new works start from scratch and still put all the focus on acquiring the best computational gains without considering accessibility, reusability nor modularity (e.g. for the study of flocking [8], crowd [24], traffic simulations [27] or autonomous navigation and path planning algorithms [4]).

Before 2011, most of those works were based on an approach that consists in executing completely the model on the GPU (called *all-in-GPU* here). This approach is useful when the main objective is only to accelerate the simulation's execution but, from a software engineering point of view, this approach is not satisfying because all the development efforts are lost due to the specificities of GPU programming: The optimization which is required eventually produces code that cannot be easily understood nor reused outside of the scope of a particular experiment.

That is the reason why hybrid approaches, that is sharing the execution of the MABS between the CPU and the GPU, begin to appear in 2011. Even if they are by design less efficient than all-in-GPU implementations, they exhibit two main advantages [9]: (1) They allow to implement more complex MABS because they do not longer need to be entirely compliant with GPGPU (see e.g. [13]) and (2) they naturally promote modularity and thus ease the development process thanks to a more concrete separation of concerns, for instance by enforcing the distinction between the agent model and the environment model (see e.g. [14,18]).

2.2 The Need for a Methodology for Porting MABS on GPU

Not surprisingly, research works that address software engineering issues in the scope of GPGPU-based MABS are now all based on an hybrid approach. However, it is worth noting that most of them rely on hiding the use of GPGPU through predefined programming languages or interfaces which are based on specific agent and environment models (e.g. [22]). Even though they represent concrete solutions for easing the use of GPGPU for MABS, such approaches cannot take into account the wide variety of MABS which can be conceived because they rely on predefined software structures and conceptual models.

[2] Compute Unified Device Architecture, https://developer.nvidia.com/what-cuda.

[3] Open Computing Language, http://www.khronos.org/opencl.

Consequently, instead of hiding GPGPU, we here argue on the idea that it would be interesting to provide the MABS community with a methodology that would concretely help to adapt and implement a MABS model using directly GPU programming. This would allow to take into account a largest number of models because such an approach would not rely on a predefined agent model and implementation. Therefore defining such a methodology is one of our main long-term goals.

3 Experimenting GPU Environmental Delegation

In this section, the GPU environmental delegation principle (*GPU delegation* for short) is introduced. Then, all the works we have done using this approach are described and their outcomes presented from both a performance and a practical perspective. Then, we show that the design guideline promoted by GPU delegation can be a starting point for the definition of a methodology dedicated to the modeling and the developing of MABS models in a GPGPU context.

3.1 The GPU Environmental Delegation Principle

Proposed in [14], the GPU environmental delegation principle is based on the fact that it is very difficult to deport the entire MABS model on graphics cards.

Inspired by an Agent-Oriented Software Engineering (AOSE) trend which consists in using the environment as a first class abstraction in MAS [1,28], GPU delegation has to be related to other research works that reify parts of the agents' computations in external structures. Examples of this trend are EASS (Environment As Active Support for Simulation) [2], IODA (Interaction Oriented Design of Agent simulations) [11], the environment-centered approach for MABS proposed in [19] and artifact approach [21].

GPU delegation thus relies on an hybrid approach which divides the execution of the MAS model between the CPU and the GPU. Especially, this principle consists in making a clear separation between the agent behaviors, managed by the CPU, and environmental dynamics, handled by the GPU. To this end, one major idea underlying this principle is to identify agent computations which can be transformed into environmental dynamics and thus implemented into GPU modules (called kernel, these modules contain the computations executed on the GPU). Originally, the GPU Environmental Delegation Principle was first stated as follows [14]: *Any agent perception computation not involving the agent's state could be translated to an endogenous dynamic of the environment, and thus considered as a potential GPU environment module.*

So, the two main design guidelines underlying this principle are: (1) Compliant environmental dynamics must be transformed into GPU modules and (2) agent computations which are compliant with the previously stated criterion should be translated in environmental dynamics and thus implemented as GPU modules.

3.2 GPU Delegation Case Studies

Multi-Level Emergence. [14] GPU delegation was first used for developing a model of Multi-Level Emergence (MLE) of complex structures inspired by [3]. This very simple model relies on a unique behavior which allows to generate complex structures which repeat in a fractal way[4]. The corresponding agent behavior is extremely simple and based on the reaction of agents to pheromones. So, in this work, two GPU modules dedicated to the perception and the spread of pheromones were proposed.

From a performance point of view, this experimentation shows that the use of GPU modules have decreased the overall execution time for each time step of the simulation. So, an acceleration gain up to x5 have been achieved, according to the hardware configuration. Figure 1 summarizes the coefficient gains obtained for three size of environment[5]: 500, 1000 and 1600.

From a practical perspective, this work had two objectives which were successfully fulfilled [14]: (1) Keeping the programming accessibility of the agent model in a GPU context (the original agent model was not modified) and (2) being able to scale up both the number of agents and the size of the environment. Moreover, this experiment showed that GPU delegation suggests a fine grained and modular approach for designing and integrating GPU modules which eases the development and maintenance tasks. Especially, this experiment shows that GPU delegation allows to tackle the genericness issue by promoting the reusability of the created GPU modules because, as they represent endogenous environmental dynamics, they deal with computations which are entirely independent from agent behavioral models. So, they could be easily reused in other contexts.

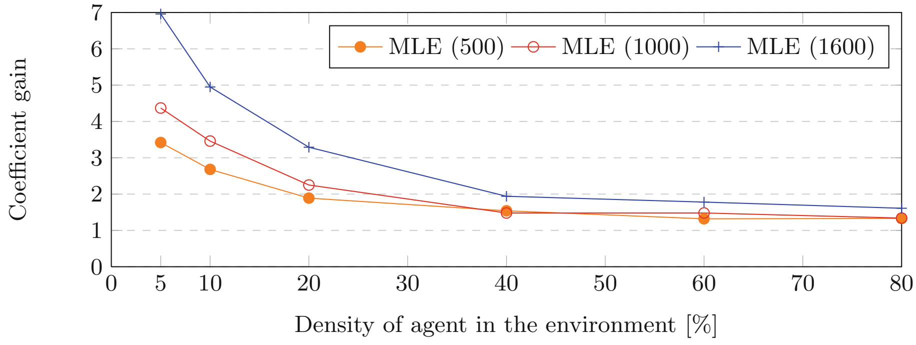

Fig. 1. Performance gains between CPU and CPU+GPU versions

Reynolds's Boids [9]. To further trial GPU delegation, a second experimentation was proposed in [9] and described how the principle has been used and eventually modified to implement a classic MABS, namely Reynolds's Boids[6].

[4] See http://www.lirmm.fr/~fmichel/mle for some videos of this experiment.

[5] In all these experiments, the environment is as a 2D square grid discretized in cells.

[6] See www.lirmm.fr/~hermellin/Website/Reynolds_Boids_With_TurtleKit.html.

In this model, each agent moves by taking into account the orientation and the speed of the others. So, applying GPU delegation, a GPU module dedicated to the computation of average orientations in each part of the environment was proposed, thus freeing the agents from doing the computation related to this perception.

To achieve this, it is worth noting that the original GPU delegation criterion had to be modified and extended so that it can be applied on this use case. Indeed, there was no agent computation fulfilling the original criterion but this experiment showed that it was in fact possible to apply GPU delegation after a slight modification of the underlying criterion. So, this latter evolved and is now stated as follows: *Any agent perception computation not **modifying** the agent's state could be translated into an environmental dynamics computed by the GPU.*

From a performance point of view, the experiment confirmed that, using GPU delegation, it is possible to increase both the size of the environment and the number of agents while obtaining an interesting speedup (up to 25 % according to the chosen parameters: Size of the environment and population density). Figure 2 summarizes the coefficient gains obtained for two size of environment: 256 and 512.

Apart from the extension of the GPU delegation criterion, from a practical perspective, this experiment also shows that the resulting agent implementation was easier to understand, thanks to the delegation of computations which are not part of the agents' deliberation.

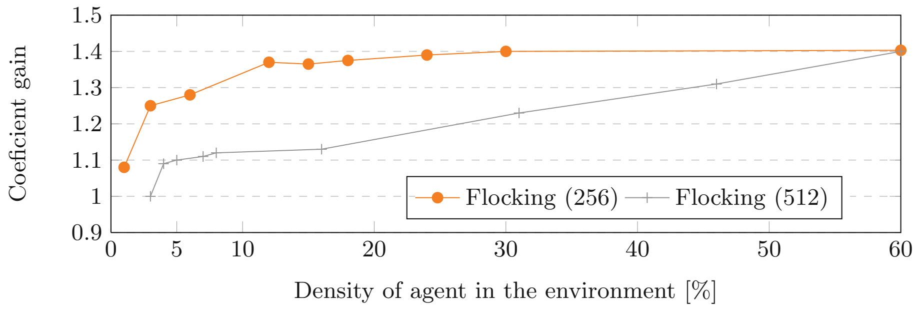

Fig. 2. Performance gains between CPU and CPU+GPU versions

Game of Life, Schelling's segregation, Fire and DLA [10]. To gain more experience on the benefits and limits of GPU delegation, it has been used to develop four different MABS simultaneously. So, [10] describes the adaption and implementation of four models: Conway's Game of Life, Schelling's segregation, Fire and DLA (diffusion-limited aggregation), the last two being taken from the NetLogo library [26][7].

[7] Detailed descriptions and computation kernels are available online: http://www.lirmm.fr/~hermellin/Website/GPGPU_MABS_Methodology.html.

In this work, three GPU modules have been created. The first one, for the Game of Life model, consists in computing for each cell of the environment its state for the next step of the simulation, according to the Game of Life rules. The second one, for Schelling's segregation, computes the happiness of each agent at each time step. The third one, for the Fire model, computes the heat diffusion in the environment. Finally, the DLA model reuses the GPU module from Game of Life (only the data structure sent to the GPU have been adapted to fit with this module) to search for the nearest neighbors in the vicinity of each agent.

From a performance point of view, Fig. 3 shows that performance gains vary significantly depending on the simulated model and the size of the environment: The gain can reach x14 but is more likely between x2 and x5.

From a practical perspective, this experiment highlighted again that GPU delegation promotes reusability by producing generic GPU modules which could be used across different MABS. Moreover, thanks to its modularity, the simplicity of the defined GPU modules shows that this approach offers a good accessibility to GPGPU because GPU delegation eventually produces simple kernels (a few lines of code are required in most cases).

Overall, beyond performance gains, this last experiment has confirmed the advantages of the approach considering software engineering aspects such as accessibility, reusability and genericity. Still, it also highlights some open questions such as the difficulty of implementation for a new GPGPU user or the types of models on which GPU delegation could be applied.

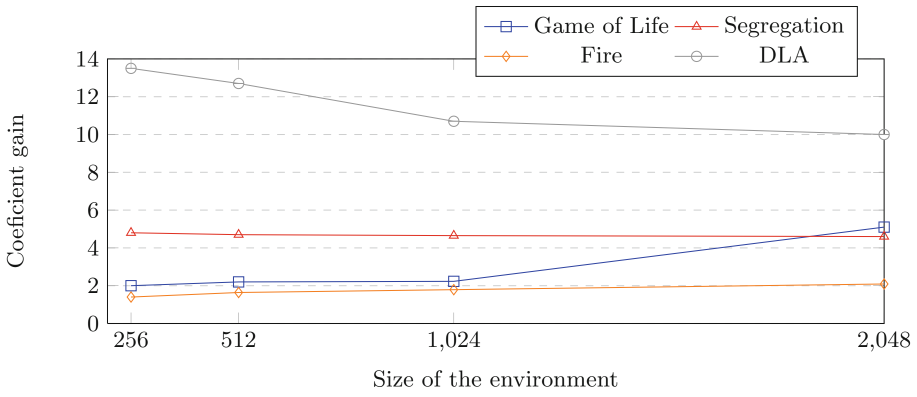

Fig. 3. Performance gains between CPU and CPU+GPU versions

4 Highlighting a Recurrent Development Pattern

From a software engineering perspective, the main objective of GPU delegation is to offer a generic approach which can be applied on a wide variety of MABS models and eases the use of GPGPU. Especially, as previously mentioned, our goal is to capitalize on our experiments to eventually produce a methodology which could be used on any kind of MABS. So, in this section, we present the different parts of the development process which we have been found as recurrent during our experiments. This enumeration is thus a first step toward a comprehensive methodology relying on GPU delegation.

With respect to the workflow that we have applied for adapting and developing the previous MABS, applying GPU delegation on a model can be divided into four main phases. The first step consists in identifying in the model all the computations which are required by the agents' behaviors or the environmental dynamics. Once these computations are referenced, the second step consists in selecting which ones could be modeled and implemented as an environmental dynamics according to the GPU delegation criterion. The third step relies on analyzing the selected computations for choosing which ones could bring interesting performance gains once translated into GPU modules. Finally, the fourth step consists in applying GPU delegation on the identified computations for developing the related GPU modules. Figure 4 summarizes this development pattern.

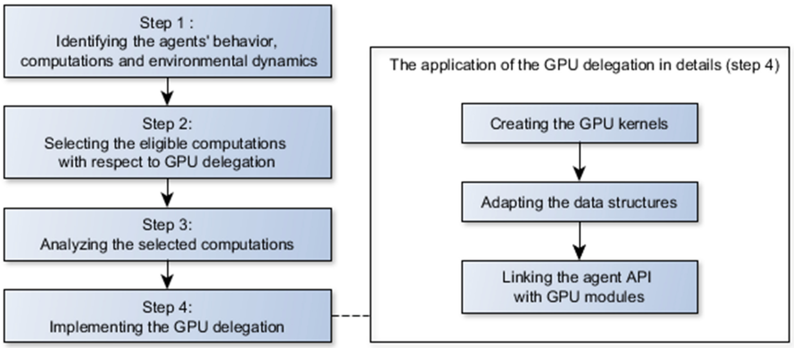
Step 1 :
Identifying the agents' behavior, computations and environmental dynamics
Step 2:
Selecting the eligible computations with respect to GPU delegation
Step 3:
Analyzing the selected computations
Step 4:
Implementing the GPU delegation
The application of the GPU delegation in details (step 4)
Creating the GPU kernels
Adapting the data structures
Linking the agent API with GPU modules

Fig. 4. Recurrent steps for the implementation of the GPU delegation

Step 1: Identifying. This phase of identification consists in decomposing the agents' behaviors to individualize as much as possible all the computations that they require. For instance, each perception process should be linked with a reified computation. The same approach holds for the environmental dynamics which are present in the model. The more the model is decomposed in simple computations, the more GPU delegation could be then successfully applied.

Step 2: Selecting the Compatible Computations. The selection of computations is an essential step because it relies on deciding which ones could benefit from GPGPU. If no part of the model is compliant with the principle, it is therefore useless to go further because, in such a case, the gains brought by GPGPU could be insignificant or even negative [13].

So, for now, in our experiments the selection of computations has been made according to the following workflow and specific criteria:

1. Considering the environment:
   - If no computations or environmental dynamics have been identified in step 1, proceed to the agents' behaviors.
   - If the environment is discretized into a 2D grid and contains computations or environmental dynamics which are applied on a large part of the cells, the delegation of these computations to GPU modules could be possible and highly relevant. Indeed, in those cases, the computations are independent from the agent's states and thus agree to the GPU delegation principle (see Sect. 3.1).
2. Considering the agents' behaviors:
   - If computations made within the behavior of the agent do not involve nor modify the agent's states, they could be translated into environmental dynamics.

Step 3: Analyzing the Selected Computations. Before applying GPU delegation on the selected computations, one must consider the performance benefits that these computations can bring. So, it is necessary to verify that (1) computations are performed by a large number of agents or applied on a lot of cells (for the environment). Because of the very high data transfer costs, if computations are rarely used, triggering a GPU computation could be not efficient even if their are compatible with the principle. Moreover, (2) the data structures used by these computations must be independent and adapted to the GPU architecture. If this is not the case, one has to check that the data can be adapted. Indeed, if the data do not match the highly specialized GPU architecture, this will impact the overall performance of the model (see [5] for more information on this aspect).

Step 4: Applying GPU Delegation. Implementing GPU delegation can be divided into three parts for each selected computation:

1. Creating the GPU kernel;
2. Adapting the data structures;
3. Linking the agent API with the GPU module.

(1) Applying GPU delegation starts with the creation of the GPU kernel, that is the GPU programming version of the selected computation. Thanks to the decomposition which have been done in the identifying step, little GPGPU knowledge is required and the produced kernels are easy to implement through a few lines of code (see e.g. [10]). (2) Then, the data structures need to be adapted to the new GPU module. This adaptation is based on the nature of both the computations and the environment model (arrays fitting the discretization of the environment are mostly used). Finally, these new elements must be integrated and linked with the CPU part of the model. So, new functions must be created to allow the agents and the environment to collect and use the data computed by the GPU module.

5 Summary and Future Works

Using GPGPU in the context of MABS remains difficult and the spreading of this technology in the MABS community is still limited, mainly because of accessibility and reusability issues.

In this context, this paper has proposed an overview of several case studies on using the GPU delegation principle for adapting MABS models to GPU programming and discussed related practical results. Especially, our experiments showed that this approach is an original and relevant solution toward the direct use of GPU programming for developing MABS. Notably, GPU delegation not only allows to obtain performance gains but also embeds a software engineering perspective that promotes GPGPU accessibility together with the resusability and genericity of the produced software modules.

This paper has then presented how the workflow which has been used in our experiments could be generalized. To this end, we described all the steps that we followed for modeling and adapting the models used in our experiments so that they can benefit from GPU programming. The underlying idea of this work is to eventually define a comprehensive methodology relying on GPU delegation. From a higher perspective, we argue on the idea that such a contribution would be crucial because it would allow to handle a wider range of models and thus contribute to the spreading of the GPGPU technology in the MABS community.

To this end, our next objective is thus to formalize the different steps and criteria which have been identified and presented in this paper to define a first version of such a methodology.

References

1. Weyns, D., Michel, F. (eds.): E4MAS 2014. LNCS (LNAI), vol. 9068. Springer, Heidelberg (2015)
2. Badeig, F., Balbo, F., Pinson, S.: A contextual environment approach for multi-agent-based simulation. In: ICAART 2010, 2nd International Conference on Agents and Artificial Intelligence, pp. 212–217, Spain, June 2010
3. Beurier, G., Simonin, O., Ferber, J.: Model and simulation of multi-level emergence. In: ISSPIT 2002, April 2008
4. Bleiweiss, A.: Multi agent navigation on the GPU. In: Games Developpement Conference (2009)
5. Bourgoin, M., Chailloux, E., Lamotte, J.-L.: Efficient abstractions for GPGPU programming. Int. J. Parallel Program. **42**(4), 583–600 (2014)
6. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Skadron, K.: A performance study of general-purpose applications on graphics processors using CUDA. J. Parallel Distrib. Comput. **68**(10), 1370–1380 (2008)
7. D'Souza, R.M., Lysenko, M., Rahmani, K.: SugarScape on steroids: simulating over a million agents atinteractive rates. In: Proceedings of Agent 2007 Conference (2007)
8. Erra, U., Frola, B., Scarano, V., Couzin, I.: An efficient GPU implementation for large scale individual-based simulation of collective behavior. High Perform. Comput. Syst. Biol. **2009**, 51–58 (2009)

9. Hermellin, E., Michel, F.: GPU environmental delegation of agent perceptions: application to reynolds' s boids. In: Gaudou, B., et al. (eds.) MABS 2015. LNCS, vol. 9568, pp. 71–86. Springer, Heidelberg (2016)
10. Hermellin, E., Michel, F.: Toward a methodology for developing MABS using GPU programming. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, AAMAS, Singapore (2016)
11. Kubera, Y., Mathieu, P., Picault, S.: IODA: an interaction-oriented approach for multi-agent based simulations. Auton. Agent Multi-Agent Syst. **23**(3), 303–343 (2011)
12. Laville, G., Mazouzi, K., Lang, C., Marilleau, N., Herrmann, B., Philippe, L.: MCMAS: A toolkit to benefit from many-core architecure in agent-based simulation. In: an Mey, D., et al. (eds.) Euro-Par 2013. LNCS, vol. 8374, pp. 544–554. Springer, Heidelberg (2014)
13. Laville, G., Mazouzi, K., Lang, C., Marilleau, N., Philippe, L.: Using GPU for multi-agent multi-scale simulations. In: Omatu, S., Paz Santana, J.F., González, S.R., Molina, J.M., Bernardos, A.M., Rodríguez, J.M.C. (eds.) Distributed Computing and Artificial Intelligence. AISC, vol. 151, pp. 197–204. Springer, Heidelberg (2012)
14. Michel, F.: Translating agent perception computations into environmental processes in multi-agent-based simulations: a means for integrating graphics processing unit programming within usual agent-based simulation platforms. Syst. Res. Behav. Sci. **30**(6), 703–715 (2013)
15. Michel, F., Ferber, J., Drogoul, A.: Multi-agent systems and simulation: a survey from theagents community's perspective. In: Uhrmacher, A., Weyns, D., (eds.) Multi-Agent Systems: Simulation and Applications, Computational Analysis, Synthesis, and Design of Dynamic Systems, pp. 3–52. CRC Press - Taylor & Francis, June 2009
16. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Comput. Graph. Forum **26**(1), 80–113 (2007)
17. Parry, H., Bithell, M.: Large scale agent-based modelling: a review and guidelines for model scaling. In: Heppenstall, A.J., Crooks, A.T., See, L.M., Batty, M. (eds.) Agent-Based Models of Geographical Systems, pp. 271–308. Springer, Heidelberg (2012)
18. Pavlov, R., Müller, J.P.: Multi-agent systems meet GPU: deploying agent-based architectures on graphics processors. In: Camarinha-Matos, L.M., Tomic, S., Graça, P. (eds.) DoCEIS 2013. IFIP AICT, vol. 394, pp. 115–122. Springer, Heidelberg (2013)
19. Payet, D., Courdier, R., Sébastien, N., Ralambondrainy, T.: Environment as support for simplification, reuse and integration of processes in spatial MAS. In: Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration - Heuristic Systems Engineering, 16–18 September, Waikoloa, Hawaii, USA, pp. 127–131. IEEE Systems, Man, and Cybernetics Society (2006)
20. Perumalla, K.S., Aaby, B.G.: Data parallel execution challenges and runtime performance of agent simulations on GPUs. In: Proceedings of the 2008 Spring Simulation Multiconference, pp. 116–123 (2008)
21. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Auton. Agent Multi-Agent Syst. **23**(2), 158–192 (2011)

22. Richmond, P., Coakley, S., Romano, D.M.: A high performance agent based modelling framework on graphics card hardware with CUDA. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009, vol. 2, pp. 1125–1126. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009)
23. Richmond, P., Romano, D.M.: Agent based GPU, a real-time 3d simulation and interactive visualisation framework for massive agent based modelling on the GPU. In: Proceedings International Workshop on Super Visualisation, IWSV 2008 (2008)
24. Richmond, P., Romano, D.M.: A high performance framework for agent based pedestrian dynamics on GPU hardware. In: European Simulation and Modelling (2011)
25. Richmond, P., Walker, D., Coakley, S., Romano, D.M.: High performance cellular level agent-based simulation with FLAME for the GPU. Briefings Bioinform. **11**(3), 334–347 (2010)
26. Sklar, E.: NetLogo, a multi-agent simulation environment. Artif. Life **13**(3), 303–311 (2007)
27. Strippgen, D., Nagel, K.: Multi-agent traffic simulation with CUDA. In: High Performance Computing Simulation, HPCS 2009, pp. 106–114, June 2009
28. Weyns, D., Van Dyke Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems state-of-the-art and research challenges. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS (LNAI), vol. 3374, pp. 1–47. Springer, Heidelberg (2005)