Proceedings Paper:
Alzahrani, E., Richmond, P. orcid.org/0000-0002-4657-5518 and Simons, A.J.H. orcid.org/0000-0002-5925-7148 (2018) A formula-driven scalable benchmark model for ABM, applied to FLAME GPU. In: Euro-Par 2017: Parallel Processing Workshops. Euro-Par 2017 International Workshops, 28-29 Aug 2017, Santiago de Compostela, Spain. Lecture Notes in Computer Science book series (LNCS) . Springer , pp. 703-714. ISBN 9783319751771

https://doi.org/10.1007/978-3-319-75178-8_56

A formula-driven scalable benchmark model for ABM, applied to FLAME GPU

Eidah Alzahrani[1,2], Paul Richmond[1], Anthony J H Simons[1]

[1] Department of Computer Science, The University of Sheffield, United Kingdom
[2] Al Baha University, Saudi Arabia

Abstract. Agent Based Modelling (ABM) systems have become a popular technique for describing complex and dynamic systems. ABM is the simulation of intelligent agents and how these agents communicate with each other within the model. The growing number of agent-based applications in the simulation and AI fields led to an increase in the number of studies that focused on evaluating modelling capabilities of these applications. Observing system performance and how applications behave during increases in population size is the main factor for benchmarking in most of these studies. System scalability is not the only issue that may affect the overall performance, but there are some issues that need to be dealt with to create a standard benchmark model that meets all ABM criteria. This paper presents a new benchmark model and benchmarks the performance characteristics of the FLAME GPU simulator as an example of a parallel framework for ABM. The aim of this model is to provide parameters to easily measure the following elements: system scalability, system homogeneity, and the ability to handle increases in the level of agent communications and model complexity. Results show that FLAME GPU demonstrates near linear scalability when increasing population size and when reducing homogeneity. The benchmark also shows a negative correlation between increasing the communication complexity between agents and execution time. The results create a baseline for improving the performance of FLAME GPU and allow the simulator to be contrasted with other multi-agent simulators.

Keywords: Agent Based Modelling , Benchmarking, Multi-Agent systems

1 Introduction

Agent-based modelling (ABM) systems (also known as multi-agent systems) have become a popular technique to study complex systems in various domains, such as biology, social sciences and business complexity. ABM can be defined as a modelling paradigm used to simulate the actions and reactions of individual entities and to measure their effects on the whole system. Many phenomena, even complex ones, can be described as systems of autonomous agents following a number of rules to communicate with each other [1].

According to Macal and North [1], the structure of an agent-based model is based on three elements: 1) the number of agents, their attributes and behaviours; 2) the agents relationships and the mechanisms with which they interact with others; and 3) each agent's environment, the actions and reactions of the agent with respect to its environment and other agents. By identifying and programming these elements, a model designer can easily create an ABM that simulates reality.

There are a number of popular agent-based modelling and simulation frameworks that are used to build models such as Swarm, NetLogo, Repast and MASON. The limitations of scalability and performance in these systems prevent modellers from simulating complex systems at very large scales. This is because some of these frameworks were designed to be run on a single CPU architecture and some of them cannot deal with a large number of agents within one model. For this reason, a number of platforms and simulators were implemented to deal with such systems. Repast HPC [11], D-Mason and FLAME GPU [26] are examples of these kinds of platforms that use parallel and distributed approaches to run simulations.

There have been several studies in the literature reporting computational performance in most ABM frameworks [2–4] for specific models. Varying the population size to measure system scalability is the most common benchmark. A benchmarking process is an excellent method to discover the characteristics of simulator performance, but unfortunately, so far there is no standard method to benchmark simulation tools. Thus there is a need to design a benchmark model that meets complexity and scalability standards. The OpenAB community[3]summarised a number of criteria that may affect the performance as follows:

- Arithmetic intensity: the computational complexity of an agent or population.
- Scale: varying population size.
- Model memory: the internal memory requirements of an agent or population.
- Inter-connectivity: the level of communication between agents.
- Homogeneity: divergence of behaviour within an agent or population.

This paper proposes a benchmark model that allows each of these criteria to be tested and we have implemented this model in FLAME GPU. The main contribution of this paper is creating a benchmark model that can be a standard to measure the execution efficiency of the existing ABM systems. This new model will be able to examine the following elements: system scalability, system homogeneity, and the ability to handle an increase in the level of agents communications and agents internal memories. The results will give insight into the performance characteristics of simulations and provide a baseline for which to measure simulator improvements.

[1] http://www.openab.org/.
[2] http://ccl.northwestern.edu/netlogo/models/community/Sugarscape
[3] http://flame.ac.uk/

2 Related Work

Numerous ABMs have been used to address a number of issues such as testing and analysing simulation tools and comparing ABM platforms, and they have been used as teaching tools for modelling real systems. This section reviews some of these models and their purposes.

Railsback et al. [14] proposed a simple model called StupidModel that can be easily implemented on any ABM platform. This model contains a number of versions to increase simulation complexity, starting from moving agents to a full predator-prey model. StupidModel was developed to be a teaching model for ABM platforms such as NetLogo and Swarm. It is also used as a benchmark model to compare modelling capabilities and performance between several ABM platform [4, 6, 10, 13] .

Predator-prey is the most commonly used model in the field of ABM and simulation. Developed by Alfred Lotka (1925) and Vito Volterra (1926), it is based on two differential equations to describe the dynamics of predator-prey behaviour. The basic rules of predator-prey in ABM can be summarised as follows: 1) two types of populations represent prey and predator agents; 2) the prey population will increase by moving to food resources and decrease by being eaten by the predators; 3) the predator population will increase by eating the prey and will decrease by starvation; and 4) both populations are moving randomly and following simple rules to communicate with the environment and with each other.

Several studies have reported comparisons of execution efficiencies between ABM platforms using predator-prey models [2, 26]. Execution efficiencies have also been used as a benchmark to show the modelling ability of Repast Simphony [11] and by Borshchev and Filippov [16] to compare three approaches to simulation modelling: System Dynamics, Discrete Events and ABM.

The Sugarscape model is an artificial society model presented by Epstein and Axtell in their book *Growing Artificial Societies: Social Science from the Bottom Up* [17]. This model was replicated by several ABM platforms such as NetLogo[4], MASON [3] and Repast [18]. Agents in the basic Sugarscape model follow very simple rules. They move towards deserted areas with high levels of sugar resources. The Sugarscape Wealth Distribution model, as described by Epstein and Axtell, has more complexity in the relations between agents.

Boids is an artificial life model developed by Craig Reynolds [19, 21] that describes the behaviour of flocking of fish or birds. According to Reynolds (2001), flocking is an example of emergence, by which the interactions of simple local rules produce a complex global behaviour. There are three simple steering behaviours that an agent in the Boids model can follow: 1) alignment, which is steering towards the average heading of nearby neighbours; 2) separation, steering to avoid crowding nearest neighbours; and 3) cohesion, steering to move toward the average position of the immediate flockmates [25]. Flocking models have been used widely to measure the modelling ability of some ABM platforms [20, 21, 26, 27].

Rousset et al. [7] used their reference model [8] to benchmark 10 existing platforms that support parallel and distributed systems. The reference model they used is based on three main behaviours for each agent: 1) agent perception, 2) agent communication and 3) agent mobility. This benchmark model is used to evaluate the ability of each platform regarding their parallelism support. A large and growing body of literature has focused on the comparison between parallel and serial execution methods to run simulations [2, 4, 5, 22–24]. All ABMs reviewed above were used as benchmarks for two purposes; to evaluate modelling capabilities of platforms and/or to make comparisons between simulators. Observing system performance and how applications behave during increases in population size is the main factor for benchmarking in most of these studies. System scalability is not the only issue that may affect the overall performance, but there are some issues that need to be dealt with to create a standard benchmark model that meets all ABM criteria.

3 The Benchmark Model

Our model is based on the concept of particle-based simulation which represents each molecule in the system as an individual entity. This entity has attributes, such as position, velocity and type of molecule. Entity movements and the reactions within the system will be computed using these attributes through methods to update system behaviour. The representation of the molecule (agent) will follow Brownian Dynamics methods, where each agent is represented as a point-like particle moving randomly in the environment.

This type of model is relevant to a wider class of ABMs. For example, both cellular models and social system models have similar behaviours, when considered from the view point of mobile agents with local interactions, birth and death and binding (combining). To make this model more complex and to meet all the criteria highlighted above, we propose a reaction-diffusion like model with different rules. Our model is able to convert formula syntax (such as A+B=C) that represents a chemical reaction to a number of mobile agents that can communicate with each other and captures important characteristics of ABM.

A simple reaction will occur when one A molecule combines with one B molecule to produce a C molecule, assuming that A+B=C represents the relationship between the three molecules. The model that resulted from the given example above contains three agents A, B, and C as follows: agent A (master agent), agent B (slave agent) and agent C (combined agent). Each of these agent specifications is defined by a set of variables and functions that help to establish the simulation. At the beginning of the simulation, agents A and B are moving randomly, and both agents are communicating with each other looking for the closest complementary agent. Agent B will send its location and then agent A will choose the closest B and replied with the ID of closest B. Once the ID of B is confirmed both agents will die and produce the new agent C.

3.1 Implementation

This section consists of three parts: 1) an overview of FLAME GPU, 2) how the benchmark model is implemented using FLAME GPU and 3) model generation. The FLAME GPU framework [9]is a template for agent-based simulation on the Graphics Processing Unit (GPU). It consists of a number of X-agents (the agent representation of an X-machine [29]) specifications. Each instance of an x-agent has its own memory that holds a set of variables. All instances of x-agents have transition functions that can read and write to their memory a start state and an end state. Agents can communicate by sending and receiving messages and their functions can read and write these messages at any time between start and end states for each agent. Creating a model using FLAME GPU is very similar to the original FLAME[5]which required writing the model specification in XML format within an X-Machine Mark-up Language(XMML) document. However, the syntax that is used to write the model in FLAME GPU uses an extended version of the FLAME XML schema. The GPUXMML extension outlines the GPU specific model description elements such as the maximum size of an agent memory [9]. This allows a formal agent specification to be transformed to optimised C-based CUDA code through GPU-specific templates.

The FLAME GPU implementation of the above example consists of three agents A, B, and C. Each agent is defined by a set of variables, transition functions, start and end states, and communication messages as shown in Table 1. The representation of agents as a state machine is shown in Fig 1. During a single iteration of the simulation, each type of agent will move from the starting state to the end state, completing each function in turn. The diagram is divided into three parts, each part showing the agent-transition functions and the communication dependency messages (green) for each agent.

At the beginning of the simulation, agents A and B are moving randomly using their move functions to update their locations during each cycle, as shown in Fig 2 Part A. Agent B will use `send_locationB` to output a locationB message holding all B information (agent ID, location, etc). Agent A after that will get all Bs locations using a `need_locationB` function that inputs the locationB message. This function will calculate the distance between A and B and then compare it with the binding radius. If the distance is less than or equal to the binding radius, the internal memory of A will be updated (the state variable will be set equal to 2, the defined value of binding(2 is the defined value of the combined state.) , and the closest ID and the closest point will be stored). The `send_bindB` function will output bindB messages holding the updated information for agent A (only messages that have the state variable equal to 2 as a function condition(*An agent function condition indicates that the agent function should only be applied to agents which meet the defined condition (which are in the correct state specified by current State [9]*)). In the next step, the `receive_bindB` function will input bindB messages to check for the closest A that is ready to combine. B's internal memory will be updated (the state variable will be set to 3(3 is the defined value of the dead state.) , and the closest ID and closest point will be stored) after finding the closest A that is ready to

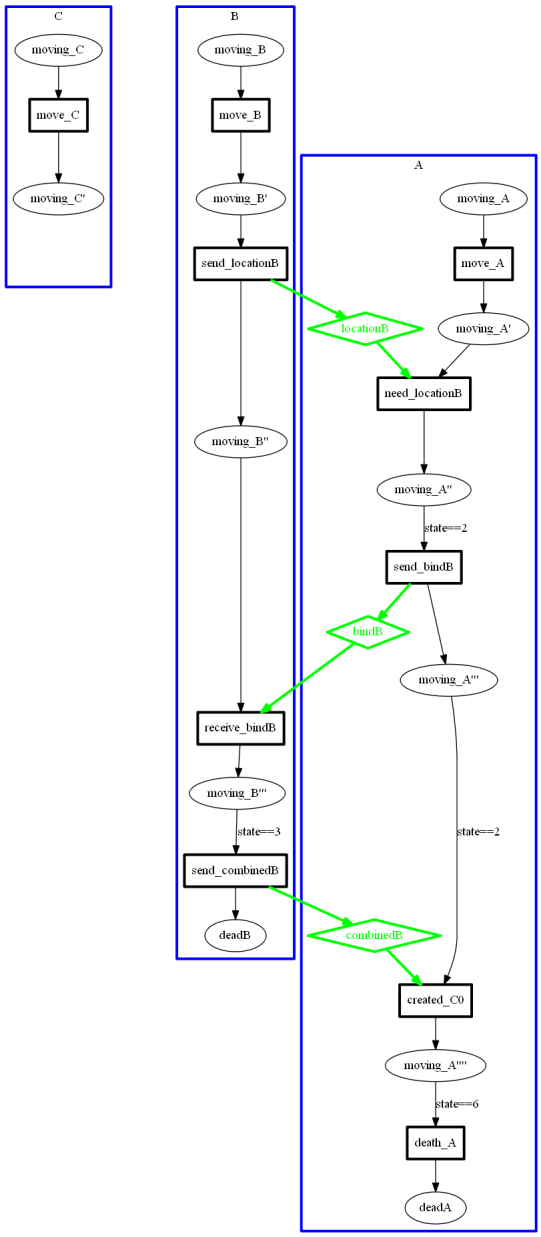

Fig. 1. State graph of the model that represents A+B=C.

combine. The `send_combinedB` function will output combinedB messages that meet the condition (the state variable is equal to 2), and the B agent will be removed from the simulation. The next function will be `created_C`. This function

Table 1. Agent specifications

Agent Type	Internal Memory	Function Name	Function Description

Master agent	Agent ID
	Agent$_{Position}$: X, Y, Z
	`Closest_id`
	`Closest_point`
	state
		1.`move_A`
		2.`need_locationB`
		3. `send_bindB`
		4. `created_C`
		5. `death_A`
			1.To update A's location
			2.Choose closest B
			3.Send request to closest B
			4.Output agent C
			5.Remove agent A from simulation

Slave agent	Agent ID
	Agent position: X,Y,Z
	`Closest_id`
	`Closest_point`
	state
		1.`move_B`
		2.`send_locationB`
		3. `receive_bindB`
		4.`send_combinedB`
			1.To update B's location
			2.Send B location
			3.Verify and choose closest A that is ready to bind.
			4.Send notification to A to combine and then remove agent B from the simulation

Combined agent	Agent ID
	Agent position: X,Y,Z
	`Closest_id`
	`Closest_point`
	state
		`move_C`	To update C's location

will input combinedB messages (only messages that meet the condition that the state is equal to 3), output agent C, and update As internal memory (the state variable will be updated to meet the next function condition). All As that meet the condition of `death_A` will be removed at this stage. A visualisation of the model after a number of iterations is shown in Fig 2 Part B.

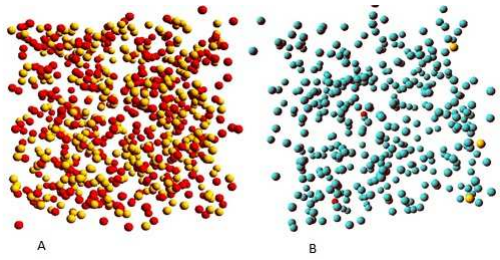

Fig. 2. Part A: Screenshot of the first iteration showing agents A (red) and B (yellow) moving randomly. Part B: Screenshot after 100 iterations showing agents C (blue) moving randomly and two of A (red) and two of B (yellow) still moving.

To save time and effort, and to implement several chemical reactions at the same time automatically, a model generator is needed. This section presents a FLAME GPU model generator that can easily convert convert lines of formula

syntax into movement models of agents. This generator after parsing the syntaxes will output three files that are required to run a FLAME GPU model: 1) a FLAME GPU XML model (XMLModelFile.xml) file that consists of model specifications, 2) a function.c file that holds the scripted agent functions, and 3) initial values of each agent for the simulation state data which is stored in a FLAME GPU XML file (0.xml).

4 Benchmarking Results

The model generator helped to vary the model in a different way and allowed modelling of different types of chemical reaction. FLAME GPU version 1.4.3 was used for the performance benchmarking on a NVIDIA GeForce GTX 970 GPU with 1665 CUDA cores and 4 GB of memory. This section shows four different benchmarks to measure FLAME GPU framework performance.

Divergence within a population: The purpose of this benchmark is to observe the system performance when doubling the number of equations. This benchmark starts with a simple model with three types of agent, ten agent functions and three type of message and ends with more than 40 agent types, 150 agent functions, and 45 message types. Adding more equation input lines (every line contains three different types of agent) increases the execution time linearly with a value of regression $\simeq 0.9945$, as shown in Fig 3(axis x1 against axis y1). processing time increases by $\simeq 0.5$ a second with the addition of a new equation. This benchmark was implemented using an agent population of 2000 for each type of agent with the same environment size, and each simulation was performed for 100 iterations.

Divergence within an agent: This benchmark gives us the average execution time for increasing slave agent types (more chemicals per line). This experiment will increase divergence within the master agent of this line. Adding a new chemical will extend the master agent functions, and that means more functions in each layer every cycle. In FLAME GPU function layers represent the control flow of simulation processes[9]. All agent functions are executed in a sequential order to complete one iteration and by adding more functions for the same agent that will increase execution time in every iteration. This can be observed in the results in Fig 3 (axis x2 against axis y2). The processing time is increasing linearly by increasing chemicals per line given a value of the regression equal to 0.9956. This benchmark was implemented using an agent population of 2000 for each type of agent with the same environment size, and each simulation was run for 100 iterations.

Population sizes: The goal of this benchmark is to measure the ability of this model to scale to examine ABM systems scalability. The population size of each agent type starts with 4,096 agents and ends with 262,144 agents. This benchmark uses A+B=C as an example to run this experiment for 100 iterations each time. The performance of implementing our model on FLAME GPU with respect to agent population size is shown in Fig 4 with linear correlation coefficient equal to 0.9811

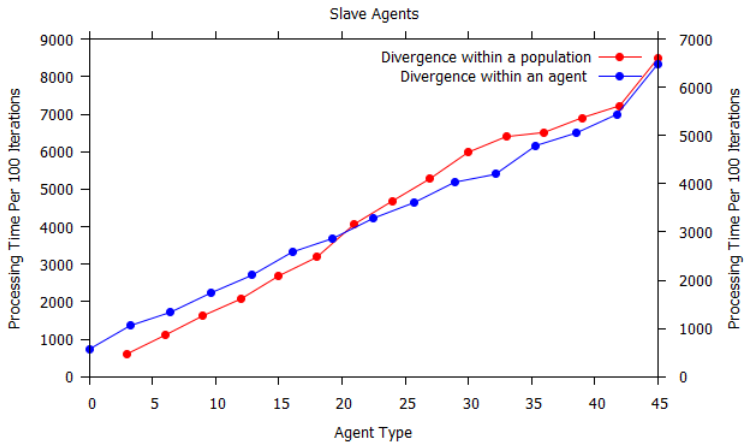

Fig. 3. Processing time of the same environment size against the type of agent that have been added at every step (appears with a red line).Processing time of the same environment size against the number of slave agents that has been added every time (appears with a blue line).

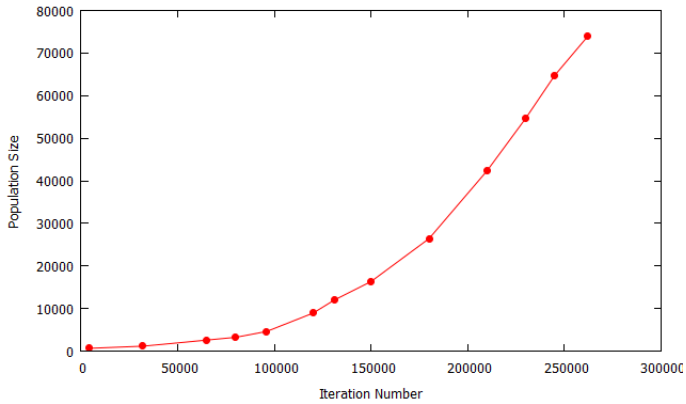

Fig. 4. Increasing population size led to increased processing time.

Level of communication and complexity: Two changes have been made to agent behaviour to slow down the simulation and add extra arithmetic intensity within agent functions: 1) decreased interaction radius and 2) decreased agent movement speed. Fig 5(axis x1 against axis y1) shows the relationship between decreasing the interaction radius and increasing processing time to produce 50 agents C from the A+B=C equation with same movement speed. This experiment allows agents to move for a longer time until reaching the needed radius, during this movement, several operations occur such as calculating agent position, sending and receiving messages between agents looking for the nearest

agent to combine with. The next experiment is shown in Fig 5(axis x2 against axis y2), which shows the relationship between slowing down the agent speed the number of iterations required to produce 50 agents. This experiment has been implemented with a constant radius and same environment size. Slowing down the movement speed allows additional operations during the simulation and this help to measure the ability of the system to handle many computational operations for a long time and how to manage using the resources.

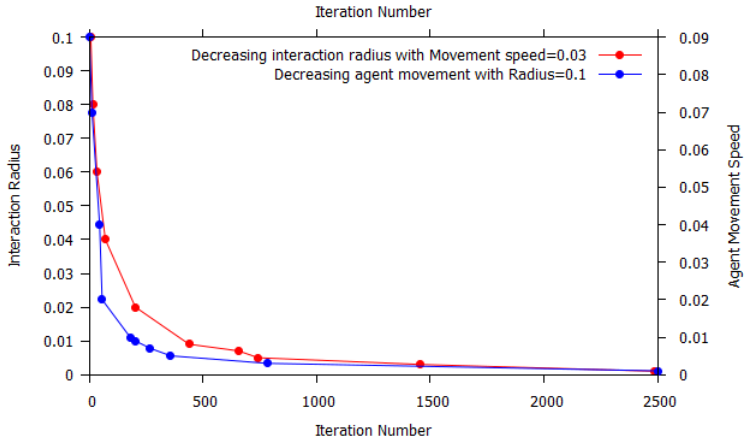

Fig. 5. Decreasing interaction radius led to increased time to produce 50 agents (appears with a red curve).Decreasing agent movement speed led to increased time to produce 50 agents (appears with a blue curve)

5 Conclusion

This paper presents the implementation of a new benchmark model using FLAME GPU. The aim of this model is to measure the following elements: system scalability, system homogeneity, and the ability to handle increases in the level of agent communications and model complexity. Unfortunately, measuring the ability to handle increases in the internal memory requirements of an agent or population was not covered by this paper. However, it will be involved in our future work.

Four benchmark experiments have been carried out, demonstrating the ability of this benchmark model to examine each element. The first two experiments focused on increasing agent and population divergence, and this led to increased the execution time due to the additional agent functions, messages and communication information that is held by these messages. The third experiment showed that we could easily scale the population size of this model to measure

the system scalability. The results showed that scaling the population size led to varying the execution time from 0.5 seconds per 100 iterations for 4069 agents till 72 seconds per 100 for 262144 agents. In the last experiment, computational complexity was added by decreasing the value of two variables that are used within agent functions to update agents behaviour. This experiment causes the model to reach a steady state at a slower rate, this allows assessment of the system capabilities.

Divergence is known to reduce performance in GPU simulations and our benchmark model confirms this. The obtained results will be used for assessing simulator improvements to achieve improved scaling with respect to divergence and better overall performance for increasing the population size. The performance results obtained indicate that our benchmark model is a suitable model to be used as an experimental tool to evaluate modelling capabilities of an ABM system if it is replicated in a suitable way.

References

1. Macal, Charles M., and Michael J. North. Tutorial on agent-based modelling and simulation. *Journal of simulation* 4.3 (2010): 151-162..
2. Fachada, Nuno, et al. Towards a standard model for research in agent-based modeling and simulation.*PeerJ Computer Science* 1 (2015): e36.
3. Bigbee, Anthony, Claudio Cioffi-Revilla, and Sean Luke. Replication of Sugarscape using MASON. *Agent-Based Approaches in Economic and Social Complex Systems IV*. Springer Japan, 2007. 183-190.
4. Lysenko, Mikola, and Roshan M. DSouza. A framework for megascale agent based model simulations on graphics processing units. *Journal of Artificial Societies and Social Simulation* 11.4 (2008): 10.
5. Dematte, Lorenzo. Parallel particle-based reaction diffusion: a GPU implementation. *Parallel and Distributed Methods in Verification, 2010 Ninth International Workshop on*, and High Performance Computational Systems Biology, Second International Workshop on. IEEE, 2010.
6. Railsback, Steven F., Steven L. Lytinen, and Stephen K. Jackson. "Agent-based simulation platforms: Review and development recommendations." *Simulation 82.9* (2006): 609-623.
7. Rousset, Alban, et al. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review 22* (2016): 27-46.
8. Rousset, Alban, et al. A survey on parallel and distributed multi-agent systems.*Padabs 2014, 2nd Workshop on Parallel and Distributed Agent-Based Simulations, in conjunction with Euro-Par 2014*. Vol. 8805. Springer, 2014.
9. Richmond, P. Flame gpu technical report and user guide. *Department of Computer Science Technical Report CS-11-03* (2011).
10. Lytinen, Steven L., and Steven F. Railsback. The evolution of agent-based simulation platforms: a review of NetLogo 5.0 and ReLogo. *Proceedings of the fourth international symposium on agent-based modeling and simulation*. 2012.
11. Tatara, Eric, et al. An indroduction to repast simphony modeling using a simple predator-prey example. *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*. 2006.

12. Grimm, Volker, and Steven F. Railsback. Individual-based modeling and ecology. Vol. 2005. Princeton: Princeton university press, 2005.
13. Standish, Russell K. Going stupid with EcoLab. *Simulation* 84.12 (2008): 611-618.
14. Railsback, Steve, Steve Lytinen, and Volker Grimm. "StupidModel and extensions: A template and teaching tool for agent-based modeling platforms." Swarm Development Group. http://condor. depaul. edu/ slytinen/abm (2005).
15. Richmond, Paul, Simon Coakley, and Daniela M. Romano. A high performance agent based modelling framework on graphics card hardware with CUDA. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
16. Borshchev, Andrei, and Alexei Filippov. From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools.*Proceedings of the 22nd international conference of the system dynamics society*. Vol. 22. 2004.
17. Epstein, Joshua M., and Robert Axtell. *Growing artificial societies: social science from the bottom up*. Brookings Institution Press, 1996.
18. Robertson, Duncan A. Agent-based models to manage the complex. *Managing Organizational Complexity: Philosophy, Theory, and Application* 24 (2005): 417-430.
19. Reynolds, Craig W. "Flocks, herds and schools: A distributed behavioral model." ACM SIGGRAPH computer graphics 21.4 (1987): 25-34.
20. Goldsby, Michael E., and Carmen M. Pancerella. Multithreaded agent-based simulation. *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. IEEE Press, 2013.
21. Reynolds, Craig. "Big fast crowds on ps3." Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames. ACM, 2006.
22. Aaby, Brandon G., Kalyan S. Perumalla, and Sudip K. Seal. Efficient simulation of agent-based models on multi-gpu and multi-core clusters. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.
23. Deissenberg, Christophe, Sander Van Der Hoog, and Herbert Dawid. EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation* 204.2 (2008): 541-552.
24. de Paiva Oliveira, Alcione, and Paul Richmond. Feasibility Study of Multi-Agent Simulation at the Cellular Level with FLAME GPU. FLAIRS Conference. 2016.
25. Reynolds, Craig. Boids: Background and update. *URL:¡ www. red3d. com/cwr/boids¿.*(Accessed 18 April 2017) (2001).
26. Richmond, Paul, and Daniela Romano. Template-driven agent-based modeling and simulation with CUDA. *GPU Computing Gems Emerald Edition, Applications of GPU Computing Series* (2011): 313-324.
27. North, Michael J., et al. *Visual agent-based model development with repast simphony*. Tech. rep., Argonne National Laboratory, 2007.
28. Macal, Charles M., and Michael J. North. Agent-based modeling and simulation. *Winter simulation conference*. Winter simulation conference, 2009.
29. Ipate, Florentin, and Mike Holcombe. "A method for refining and testing generalised machine specifications." *International journal of computer mathematics* 68.3-4 (1998): 197-219.