Benchmarking the Agent Descriptivity of Parallel Multi-Agent Simulators

Craig Shih, Caleb Yang, and Munehiro Fukuda

University of Washington, Bothell WA 98011, USA,
{cshih, yangc9, mfukuda}@uw.edu,
WWW home page: http://depts.washington.edu/dslab/MASS/index.html

Abstract. Agent-based models (ABMs) need to populate a mega number of agents over a scalable simulation space in order to handle practical problems, (e.g., metropolitan traffic simulation and nationwide epidemic prediction). Although parallel and distributed simulation have steadily addressed their computational needs, non-computing scientists still tend to use GUI-rich, easy-to-use ABM interpretive platforms. This paper intends to identify the difficulty in using the current parallel ABM simulators and to propose their future improvements. For this purpose, we surveyed different ABM applications, modeled them as seven benchmark test cases, used them to analyze the agent descriptivity of parallel ABM simulators, and evaluated their execution performance affected by the current implementations.

Keywords: agent-based models, parallel simulation, agent programmability, benchmark tests

1 Introduction

Agent-based models (ABMs) have been highlighted as micro-simulation that models microscopic events, (e.g., transport simulation involving traffic lights, constructions, and pedestrian movements) and that observes an emergent collective group behavior of many simulation entities (which are called agents). Since not all scientists are computing specialists, GUI-rich, easy-to-use ABM interpretive platforms including NetLogo[1], Repast Symphony[2], and MASON[3] have attracted them for the last 10+ years.

Emergent worldwide tensions including globalization, urbanization, population increase, and Internet dissemination require the ABM problem size to be scaled up, which increases computation time unacceptably. For instance, FluTE, an influenza epidemic simulation model [7] takes 2 hours to simulate an unmitigated epidemic in 10 million people. Scientists have noticed that parallel computing is imperative for large-scale simulations of ABMs [8]. However ABM

[1] http://ccl.northwestern.edu/netlogo/
[2] https://repast.github.io/repast_simphony.html
[3] https://cs.gmu.edu/~eclab/projects/mason/

interpretive platforms are difficult to parallelize or to speed up. In fact, NetLogo has not yet been parallelized. D-MASON[4], a distributed version of MASON has been no longer maintained for the last four years.

Contrary to these interpretive platforms, FLAME[5] and RepastHPC[6] are C/C++-based platforms that facilitate ABM parallelization over a cluster system and have successfully promoted scalable computing. EURANCE is a massively parallel agent-based simulation of European economy where FLAME populated 3,500 through to 500,000 agents [10]. RepastHPC has been tested for scalability on Blue Gene/P and achieved exascale computing on Illinois electric power transmission grid and services [26]. Despite that, as mentioned above, many scientists are still using NetLogo and Repast Symphony.

From our programming experiences with FLAME and RepastHPC [6], we observe two hurdles for non-computing specialists to clear when using these two systems: (1) tolerating limited flexibilities in programming and (2) understanding parallel-computing concepts. Based on their own design philosophy, FLAME and RepastHPC give a specific programming framework that guarantees automatic parallelization of user applications as far as they are coded in accordance with the framework. However, their learning curves are very steep. Moreover, users still need to understand underlying parallel-computing platforms such as message boards in FLAME and MPI in RepastHPC.

Given these backgrounds, we have surveyed programming styles in various ABM applications in Section 2; focused on two C++-based ABM parallel simulators in Section 3, which are RepastHPC and MASS, (the latter of which is our own simulator [6]); developed a benchmark test set based on the survey and identified challenges in current parallel ABM platforms from the viewpoint of both programmability and execution performance in Section 4; and clarified their challenges and sought for future solutions in Section 5.

2 Challenges in Agent-Based Modeling

This section surveys ABMs, examines potential parallelizations with conventional ABM platforms, and clarifies their challenges in parallelization.

2.1 ABM Applications

We are looking at ABMs in social, behavioral and economic (SBE) sciences as well as biological, ecological, and city-planning applications that are tightly coupled with SBE. Although our survey is still in progress, the following samples some ABM applications in each scientific discipline.

1. ***Social science***

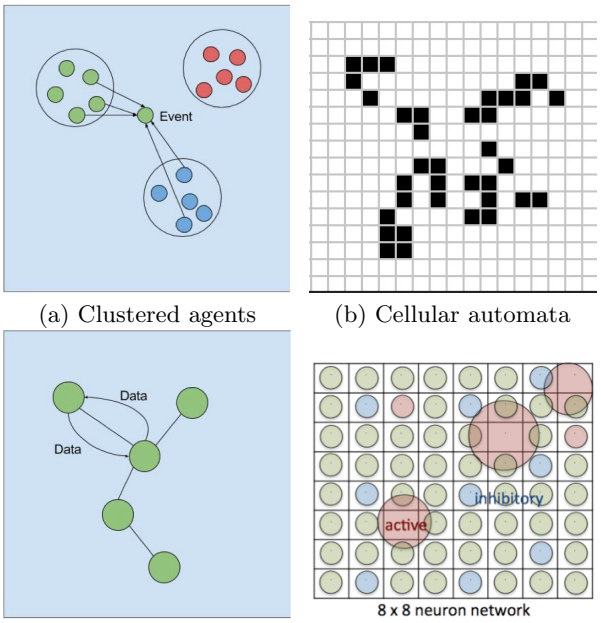
[4] https://sites.google.com/site/distributedmason/
[5] http://www.flame.ac.uk
[6] https://repast.github.io/repast_hpc.html

   (a) *Social Network Modeling:* simulates activities among and event influence to online communities [1, 9]. It can be modeled as a network of agents, each exchanging information along links with its neighbors.
2. ***Behavioral science***
   (a) *Flows:* observe the dynamics of indoor and outdoor pedestrian crowd[7] and metropolitan emergency evacuation [20, 21]. These simulations use cellular automata or observe agent movement over a 2D gird.
   (b) *Organizations:* forecast the team productivity for certain operations. For instance, Virtual Design Team (VDT) models a hierarchical software developer team that moves from one to another product phase [19].
3. ***Economic science***
   (a) *Markets:* include bank bail-in/out [18] and Internet service provider (ISP) markets [3], both observing interactions among clusters of agents. The former distinguishes three clusters: firms, banks, and households, whereas the latter creates different ISP markets where customer agents move from one to another.
   (b) *Diffusions:* are quite closer to social network modeling where a product adoption is modeled as a word-of-mouth diffusion on social networks [3].
4. ***Biological/Ecological sciences***
   (a) *Cell-level simulation:* simulates the growth of synapses [17], identifies control mechanism of granuloma formation by TB bacteria, macrophages and T-cells [25], and cell-level blood flow throughout the microvasculature network [27].
   (b) *Ecosystems:* simulate an emergent collective group behavior of artificial lives. The examples include Wa-Tor: an ecological war between sharks and fish [11] and Sugarscape: an artificial society [12], both based on agent migration over a 2D space. Conway's Game of Life, a well-known cellular automata game [14] could be considered as an artificial life.
   (c) *Epidemic simulation:* predicts a pandemic of a given disease such as dengue fever [16] and influenza [7]. They move agents from one to another community.
5. ***Urban planning***
   (a) *Transport simulation:* predicts traffic flows under a given condition. The major simulators are TRANSIMS [2] and MATSim[8], each based on cellular automata and a queuing network respectively.
   (b) *Land use simulation:* simulates agent behaviors over households, businesses, and land areas[9]. The model is generally combined with transport simulation.

Apparently, some of these applications are based on a similar computing model in terms of how agents are spawned and interact with each other over their shared space. Therefore, we have categorized them from scientific disciplines into computing models from the viewpoints of static versus dynamic agents as well as based on the structure of their shared space:

[7] http://pedsim.silmaril.org/
[8] http://www.matsim.org
[9] http://www.urbansim.com

1. ***Static agents:*** have no mobility over a simulation space.
   (a) *Clustered agents:* form one or more groups, each exchanging events with others and sharing them among its internal agents (Figure 1-(a)). Market simulation is this model.
   (b) *Cellular automata:* consider each cell as a static agent that communicates with its neighbors (Figure 1-(b)). This model includes Conway's Game of Life and TRANSIMS.
   (c) *Network of agents:* is a more generalized form of cellular automata by replacing a 2D/3D grid with a network (Figure 1-(c)). Social network modeling and economic diffusions belong to this category.
   (d) *Agents with dynamic communication:* stay static but extend its communication paths to further agents as seen in the growth of synapses (Figure 1-(d)).

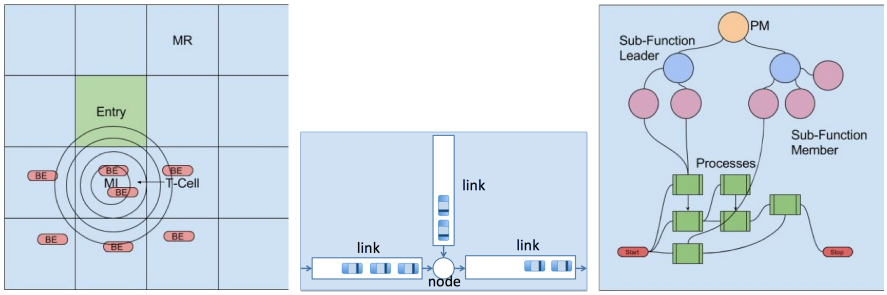


(a) Clustered agents (b) Cellular automata

(c) Network of agents (d) Dynamic communication

Fig. 1. Categorization for static agents

2. ***Dynamic agents:*** have navigational autonomy over a space.
   (a) *Agent migration over grid:* moves agents over a 2D or a 3D space as shown in Figure 2-(a). Most ABM applications in biology and ecology can be categorized in this model.
   (b) *Agent migration over network:* walks or drives agents over a network (of roads or microvasculature) as illustrated in Figure 2-(b)).
   (c) *Group migration over multi phases:* is a very special case in agent migration that corresponds to VDT. Figure 2-(c) models a developer team's transition from one to another product phase.



(a) Migration over grid (b) Migration over network (c) Group migration

Fig. 2. Categorization for dynamic agents

We understand that several more computing surveys on ABM research areas and applications have been already published[10] [4, 3, 24, 13], and thus we need to cover all these surveyed models. However, our preliminary ABM categorization shown above can serve as a useful tool for our initial review on how efficiently conventional approaches have parallelized these computing models and what challenges still lie for ABM parallelization. Below we look into conventional approaches to ABM parallelization.

2.2 Conventional Approaches

As ABM scientists have confronted issues of scaling up their model size and calibrating their models from previous runs, (which were predicted as of 2002 [15]), they began to focus on parallel computing that is effective to increase both computation speed and space. The majority of ABM platforms based on interpretive languages such as Java and Logo suffer from their slower code interpretation than native mode execution, generally by one order of magnitude. Their parallelization is normally limited to multithreading built in their language, and is thus based on the shared-memory programming paradigm. To stay in minor changes for parallelization, D-MASON, (i.e., a distributed version of MASON) uses a centralized communication server that maintains the consistency of agent migration and communication, which contributes to only spatial scalability. Therefore, interpretive ABM platforms are generally bound to inherent slow execution and multithreading.

There are several parallel and distributed ABM systems that have addressed ABM computational needs in both speed and space by implementing a native-mode execution platform over a cluster of computing nodes [23]. Among them, RepastHPC is an MPI-supported C++ system where *Context* is an execution environment that populates agents over a given *Projection* instance such as a shared grid and space. However, one of RepastHPC's drawbacks is its heavy dependence on MPI, which results in a steep learning curve for users.

Another native-mode parallel platform is FLAME. Since FLAME users write their simulation in C, for object-based programming purposes, they need to

[10] http://www2.econ.iastate.edu/tesfatsi/ace.htm

declare all agents and environment variables in XML, in a similar way to C++ header files. Although the environment variables are used to shape a simulation space, FLAME does not instantiate any actual space in memory. Instead, agents are capable of broadcasting their messages among one another through message boards, each launched at a different MPI rank. Contrary to RepastHPC, FLAME is considered as a collection of communicating, state-transitting agents statically mapped over MPI ranks. From this viewpoint, FLAME would burden ABM designers with application-level manipulation of dynamic agents.

Some mobile agents were capable of creating trees or logical networks over a cluster system. Olden recursively created new tree branches, spawned and moved child threads along the branches, and let them execute a new function [22]. WAVE enabled its interpretive agents to create new network links/nodes and to dispatch their offsprings [5]. Their drawbacks are slow execution due to thread migration, interpretive execution, and overheads incurred by logical network creation.

These circumstances motivate us to examine common issues in parallelizing ABMs by developing a benchmark test set and to address them by proposing solutions to existing parallel platforms such as RepastHPC as well as enhancing our own MASS library as a general-purpose simulation environment.

3 Parallel ABM Simulators with Agent Mobility over Virtual Space

We focus on two parallel ABM platforms, MASS and RepastHPC, both of which facilitate agent mobility and thus cover execution of not only static but also dynamic agents.

3.1 MASS Library

We have developed and released to the public the MASS (Multi-Agent Spatial Simulation) library in Java and C++. The library has two key classes: *Places* and *Agents*. *Places* is a multi-dimensional array of elements that are dynamically allocated over a cluster of multi-core computing nodes. Each element is called a *place*, is pointed to by a set of network-independent array indices, and is capable of making a parallel function call with `Places.callAll(place.functionId)`, exchanging information with any other *place*s through `exchangeAll()`, and facilitating visibility between neighboring *place*s with `exchangeBoundary()`. *Agents* are a set of execution instances that can reside on a *place*, autonomously migrate to any other *place*s with array indices through `Agent.migrate()` and `Agents.manageAll()`, and indirectly interact with other *agent*s through variables local to the current *place*. Parallelization with the MASS library uses a set of multi-threaded communicating processes that are forked over a cluster of multi-core computing nodes with libssh2 in C++ and are connected to each other through TCP sockets. Multi-threads take charge of parallel method invocation and information exchange among *place*s and *agent*s.

3.2 RepastHPC

RepastHPC is an agent-based modeling and simulation framework that is implemented in C++ on top of MPI to facilitate high-performance parallel and distributed simulation. Agents are implemented as objects (C++ classes), each identified with a unique `repast::AgentId` and maintaining its state represented by the variables in the classes. The agent behavior is described by the functions in those classes. A *Context* is used to encompass the population of agents. Each context can only contain a single type of agents. When an agent is created, it is added to a Context with `addAgent()`. When they die, they are then removed from the Context with `removeAgent()`. *Context* also takes charge of moving agents over a shared space, using `move()` and `balance()`. RepastHPC distinguishes various simulation spaces called *Projections*, including `SharedNetwork`, `SharedDiscreteSpace`, `SharedContinuousSpace`, `DiffusionLayerND`, and `ValueLayerND`, each respectively modeling a different simulation space: a 2D/3D shared discrete space, a shard contiguous space, an N-dimensional layer of diffusing values over a space, and an N-dimensional array of values accessed by agents. These spaces provide agents with *Moore2DGridQuery* and *RepastEdgeContentManager* classes to find their neighbors within a closer distance or reachable through an agent network edge. RepastHPC utilizes a dynamic discrete-event scheduler with conservative synchronization. Events are scheduled to occur at a specific tick which is also used to determine order.

4 Comparisons

This section compares MASS and RepastHPC for agent and spatial descriptivity to identify what they lack for attracting attention from non-computing users. Thereafter, our benchmark test cases examine execution overheads incurred by their implementation when they run in parallel.

4.1 Analysis of Agent and Spatial Descriptivity

Six different types of applications have been chosen from Section 2.2, simplified as a general benchmark test, and coded in MASS C++ and RepastHPC[11]: Bank bail-in/out (*Bank*) as clustered agents, Game of Life (*Life*) as cellular automata, social network (*SocialNet*) as a network of agents, BrainGrid (*Brain*) as agents with dynamic communication, Tuberculosis simulation (*TB*) as agent migration over grid, and multi-agent transport simulation (*Transport*) as agent migration over network. Note that the seventh test case, (i.e., group migration) is still in progress.

Tables 1 and 2 summarize the model descriptivity of these six test cases in terms of simulation space and agent managements respectively. In space management, while RepastHPC facilitates both multi-dimensional spaces and any graph topologies as a simulation space, MASS must manage to emulate graphs using

[11] https://bitbucket.org/mass_application_developers/

its *Places* array structure. On the other hand, RepastHPC supports only passive space for agents to store, retrieve, and share data, whereas *Places* in MASS can behave as a static collection of elements actively performing computation.

Features	Appl.	MASS	RepastHPC
Creation			
Arrays	Life, TB Brain,Bank	+ Yes, with Places	+ Yes, with SharedDiscreteSpace
Graphs	SocialNet Transport	− Mimicked by Places, using neighbors in exchangeAll()	+ Yes, with successors in the agentNetwork class
Addressing			
Space	Life, TB Brain,Bank	− Only array indices allowed	+ Yes, with SharedDiscreteSpace and SharedContinuousSpace
Graphs	SocialNet	− Mimicked by neighbors in exchangeAll()	+ Yes, with successors in the agentNetwork class
Communication			
In array	Life, TB Brain	+ Yes, with exchangeBoundary()	+ Yes, with moor2Dspace()
Over graph	SocialNet	− Mimicked by neighbors in exchangeAll()	+ Yes, with agentNetwork and DiffusionLayerND.diffusion()
Space & agents	Transport TB, Bank	+ Yes, through Place.agents and Agent.place pointers	− From agent to space only with ValueLayerND.get/setValueAt()
Computation	All	+ Yes, with Places.callAll()	− Performed by agents only. No active space concepts

Table 1. Spatial management

In agent management, RepastHPC needs user interventions to facilitate runtime agent creation and termination although it can populate agents over an entire space from the hawk's viewpoint, whose merits and demerits are opposite in MASS. Due to their different design concepts, MASS allows different classes of agents to behave on the same *Places*, whereas RepastHPC allows different *Context*s to share the same *ValueLayerND* space. As MASS cannot create a network of agents, their direct communication is not supported. On the other hand, as RepastHPC supports read-only ghost space, agents may get collided onto the same logical location, which must be handled at a user level.

In entire simulation management as summarized in Table 3, MASS focuses on hiding the underlying parallel platforms from ABM applications much more than facilitating user-friendly event scheduling and data collection. On other hand, RepastHPC exposes users to underlying parallel platforms. They must understand MPI and handle agent/data packing, transfer, and unpacking. They must also maintain agent lists in Context and examine one after another agent, which is tedious and difficult to parallelize. None of both supports strong migration.

Table 4 compares MASS C++ and RepastHPC in lines of code (LoC) necessary to describe the test cases. In general, MASS C++ can describe these six test cases in 64% through 94% of the corresponding RepastHPC's code. However, this is because RepastHPC needs to add MPI-parallel and message pack/unpack code. In fact, RepastHPC can describe agent behavior in less LoC than MASS C++ when agents are static in space.

Features	Appl.	MASS	RepastHPC
Creation			
Population	All	− Individually with Agent.map	+ Entirely with Context.addAgent()
Multiclasses	Transport TB	+ Yes, with new Places()	− One class of agents per context but ValueLayerND shared among multiple contexts
Runtime	TB	+ Yes, with Agent.spawn() and Agent.kill()	− Marked as dead at a user level, then removed by agentRemove()
Communication			
Direct	Social	− No support	+ Yes, through the agentNetwork class
Indirect	Transport TB	+ Yes, through Place.agents or Place-local variables	+ Yes, through the moore2Dspace class
Broadcast	Bank	+ Yes, by sending a financial messenger to an agent cluster	− Emulated by having a firm/bank contact each of agents in their cluster
Computation	All	+ Yes, with Agents.callAll()	+ Yes, RepastHPCAgent.play()
Migration	Transport TB		
Autonomy		+ Yes, with Agent.migrate()	+ Yes, with RepastHPCAgent.move()
Strong migration		− No, weak migration only	− No, weak migration only
Collision		+ Yes, supported by system	− No, user emulation using the moore2Dspace class

Table 2. Agent management

Features	MASS	RepastHPC
Scenario control	− main() describes an entire scenario.	+ Context registers discrete events.
Parallelization	+ Multi-processes spawned by libssh2	+ Multi-processes spawned by MPI
	+ Multithreading supported	− No multithreading for agent manipulation within each Context
	+ System-supported agent/data transfer	− User-operated data pack/unpack
	− Synchronization required for agents to access the same place	+ Synchronization eased with sequential agent manipulation within Context
Outputs	− MASS.log() collects data per process.	+ repast::DataSet::record() collects data at rank 0.
	− Both give users a burden to keep track of a specific agent.	

Table 3. Entire simulation management

In summary, there is still some room for improvement in ABM parallelization before it attracts non-computing users: (1) underlying parallel platform should be masked as observed in MASS and initialization/data collection should be centralized as observed in RepastHPC; (2) ABM code can be further simplified with a plenty of simulation features including: discrete event scheduling, contiguous simulation space, graph construction, active computation in space, and more advanced agent mobility such as strong migration and agent collision avoidance.

4.2 Execution Performance

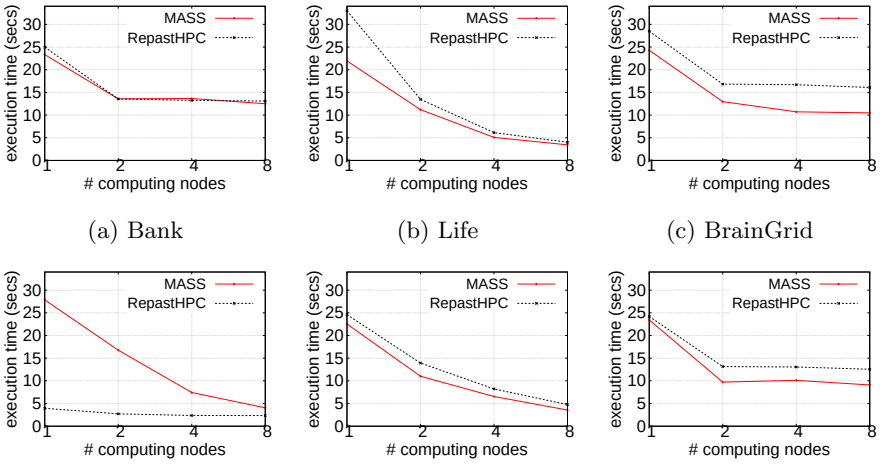
We evaluated the parallel performance of MASS and RepastHPC, using the University of Washington, Bothell's shared Linux cluster: 16 Dell Optiplex 710 desktops, each with an Intel i7-3770 Quad-Core CPU at 3.40 GHz and 16 GB RAM. Figure 3 shows the results of the six test cases. *Bank*, *Life*, *BrainGrid*, and *SocialNet* are static agents, while *Transport* and *TB* are dynamic agents. We adjusted their space size to $100\times100 \sim 1000\times1000$ and # agents to $200 \sim 162000$ so that all tests complete below 35 seconds for their sequential execution.

	Bank		Life		BrainGrid		Social Net		Transport		TB	
Code	MASS	Repast	MASS	Repast	MASS	Repast	MASS	Repast	MASS	Repast	MASS	Repast
Env/Space	302	400	107	386	120	386	114	409	344	398	424	407
Agents	227	194	223	131	467	240	332	161	185	307	143	200
Total	529	594	330	517	587	626	446	575	529	705	567	607
Ratio for entire code	0.89	1.00	0.64	1.00	0.94	1.00	0.78	1.00	0.75	1.00	0.93	1.00
Ratio for agent code	1.17	1.00	1.71	1.00	1.95	1.00	2.00	1.00	0.60	1.00	0.72	1.00

Table 4. Lines of code in MASS C++ and RepastHPC

In static agents, *Bank* gathers agents in groups and thus showed less spatial parallelism. For better performance, agents must be handled with multithreading. *Life* demonstrated their super-linear performance due to more space made available with multiple nodes. *BrainGrid* did not perform better with 4+ nodes, because its computation completed quickly with the current random initialization of neurons. *SocialNet* was executed much faster, using RepastHPC's `agentNetwork` class that supports a graph of static agents. In dynamic agents, while *Transport* showed its clear CPU scalability, *TB* suffered from unbalanced load over 4+ computing nodes, because its agents tend to come together, which should be addressed with dynamic load balancing. It is inevitable that RepastHPC performs slower than MASS (except *SocialNet*) due to its MPI-based implementation and user-level agent serialization, whereas MASS directly uses sockets and automates all the serialization.

In summary, although a cluster system facilitates more computing resources in CPUs and memory that allow a simulation space and # agents to scale up, parallel simulators need to balance initial and dynamic distribution of a space and agents over cluster nodes, and should utilize CPU cores more efficiently.



(a) Bank (b) Life (c) BrainGrid

(d) SocialNet (e) Transport (e) TB

Fig. 3. ABM execution performance

5 Conclusions

We surveyed ABM applications in SBE, biological and ecological sciences, and urban planning; categorized into seven models from the viewpoint of agent mobility; and analyzed MASS and RepastHPC for their agent and spatial descriptivity as well as measured performance overheads incurred by their implementation.

Based on our agent descriptivity analysis, we proposed that parallel ABM simulators should hide parallel-computing constructs from non-computing users, enrich simulation-space topological and computational features, and improve agent mobility as they can mimic human, animal, and moving objects. Our performance evaluation demonstrated substantial performance improvements with two cluster nodes but not always with 4+ nodes. For further scalable computation, ABM simulators should efficiently handle agents with CPU cores and to distribute a simulation space and agents uniformly over a cluster system, which balances computation and communication over computing nodes.

To pursue this research project, we will complete the VDT model as the seventh test case; fine-tune the complexity of all the seven test cases for covering more realistic agent behaviors; extend our performance evaluation to 32 ways, (i.e., 8 cluster nodes $\times$ 4 CPU cores per node); and develop agent annotations and sub-classes that predefine typical agent behavior at the system level.

References

1. Ang, C.S., Zaphiris, P.: Simulating Social Networks of Online Communities: Simulation as a Method for Social Design. In: IFIP Conference on Human-Computer Interaction - INTERACT 2009. pp. 443–456. Uppsala, Sweden (August 2009)
2. Barrett, C.L., et al.: TRANSSIMS(TRansportation ANalysis SIMulation System) Vol 0 - Overview. La-ur-99-1658, Los Alamos National Laboratory (May 1999)
3. Bohnabeau, E.: Agent-based modeling: methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences of the United States of America Vol.99(No.3), 7280–7287 (May 2002)
4. Borrill, P.L., Tesfatsion, L.: The Elgar Companion to Recent Economic Methodology, chap. Agent-Based Modeling: The Right Mathematics for the Social Sciences?, pp. 228–258. Edward Elgar Publishers (February 2011)
5. Borst, P.: The first implementation of the WAVE system for UNIX and TCP/IP computer networks. Tech report 18/92, University of Karlsruhe (December 1992)
6. Bowzer, C., Phan, B., Cohen, K., Fukuda, M.: Collision-free agent migration in spatial simulation. In: In Proc. of 11th Joint Agent-oriented Workshops in Synergy (JAWS'17). Prague, Czech (September 2017)
7. Chao, D.L., Halloran, M.E., Obenchain, V.J., Longini Jr, I.M.: FluTE, a Publicly Available Stochastic Influenza Epidemic Simulation Model. PLoS Computational Biology Vol.6(No.1), 517–527 (January 2010)
8. Chiacchio, F., PEnnisi, M., Russo, G., Motta, S., Pappalardo, F.: Agent-Based Modeling of the Immune System: NetLogo, a Promising Framework. BioMed Research International Vol.2014(No.2014) (April 2014)
9. Cicirelli, F., et al.: Edge Computing and Social Internet of Things for large-scale smart environments development. IEEE Internet of Things Journal (2017), dOI: 10.1109/JIOT.2017.2775739

10. Deissenberg, C., van der Hoog, S., Dawid, H.: Eurace: A massively parallel agent-based model of the european economy. Applied Mathematics and Computation Vol.204(No.2), 541–552 (2008)
11. Dewdney, A.K.: Computer recreations sharks and fish wage an ecological war on the toroidal planet wa-tor. Scientific American pp. 14–22 (December 1984)
12. Epstein, J., Axtell, R.: Growing artificial societies: social science from the bottom up, p. 224. Brookings Institution Press (October 1996)
13. Fortino, G., et al.: Modeling and simulating internet-of-things systems: A hybrid agent-oriented approach. Computing in Science & Engineering Vol.19(No.5), 68–76 (2017)
14. Gardner, M.: The fantastic combinations of john conway's new solitaire game "life". Scientific American Vol.223, 120–123 (October 1970)
15. Gilbert, N., Bankes, S.: Platforms and methods for agent-based modeling. Proceedings of the National Academy of Sciences of the United States of America Vol.99(No.3), 7197–7198 (May 2002)
16. Jacintho, L.F.O., Batista, A.F.M., Ruas, T., Slive, F.A.: An agent-based model for the spread of the dengue fever: A swarm platform simulation approach. In: Proc. of the 2010 Spring Simulation Multiconference, SpringSim 2010. SCS, Orland, FL (April 2010), doi: 10.1145/1878537.1878540
17. Kawasaki, F.: Accelerating large-scale simulations of coortical neuronal network development. Master's thesis, MSCSSE, University of Washington Bothell (2012)
18. Klimek, P., Poledna, S., Farmer, J.D., Thurner, S.: To bail-out or to bail-in? answers from an agent-based model. Journal of Economics Dynamics & Control Vol.50, 144–154 (January 2015)
19. Levitt, R.E.: VDT Computational Emulation Models of Organizations: State of the Art and Pracice. Technical report, Stanford University (2000), http://web.stanford.edugroup/VDT/VDT.pdf
20. Lu, W., Liu, C., Bhaduri, B.: Agent-based Large-Scale Emergency Evacu-ation Using Real-Time Open Government Data. In: Proc. of Workshops on Big Data and Urban Informatics. Chicaco, IL (August 2014)
21. Nuria, P., Allbeck, J.M., Badler, N.I.: Controlling individual agents in high-density crowd simulation. In: Proc. of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer Animation. Eurographics Association (2017)
22. Rogers, A., et al.: Supporting Dyanmic Data Structures on Distributed-Memory Machines. TOPLAS Vol.17(No.2), 233–263 (March 1995)
23. Rousset, A., Herrmann, B., Lang, C., Philippe, L.: A survey on para-llel and distributed multi-agent systems for high performance computing simulations. Computer Science Review Vol.22, 27–46 (November 2016)
24. Savaglio, C., et al.: Agent-based computing in the internet of things: A survey. In: Proc. of the 11th Int'l Symposium on Intelligent and Distributed Computing. pp. 307–320. Springer, Belgrade, Serbia (October 2017)
25. Segovia-Juarez, J.L., Ganguli, S., Kirschner, D.: Identifying control mechanisms of granuloma formation during M. tuberculosis infection using an agent-based model. Journal of Theoretical Biology Vol.231(No.3), 357–376 (December 2004)
26. Suresh, S., Gutmann, M.P.: Rebuilding the Mosaic: Fostering Research in Social, Behavioral, and Economic Sciences at the National Science Foundation in the Nexst Decade. Nsf 11-086, National Science Foundation, Directorate for Social, Behavioral and Economic Sciences, Arlington, VA USA (2011)
27. Thorne, B.C., Bailey, A.M.: Multi-cell Agent-based Simulation of the Microvasculature to Study the Dynamics of Circulating Inflammatory Cell Trafficking. Annals of Biomedical Engineering Vol.35(No.6), 916–936 (June 2007)