Benchmarking and Evaluating ABM Parallel-Programming Features in Social, Behavioral and Economic Sciences

Craig Shih

A Capstone Project
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2018

Committee:

Munehiro Fukuda, Committee Chair

Erika Parsons, Committee Member

Michael Stiber, Committee Member

Program Authorized to Offer Degree:
Department of Computing and Software Systems

University of Washington

Abstract

Benchmarking and Evaluating ABM Parallel-Programming Features in Social, Behavioral and Economic Sciences

Craig Shih

Chair of the Supervisory Committee:
Professor Munehiro Fukuda
Computing & Software Systems

Agent-based models (ABMs) have been highlighted in social, behavioral, economical, and environmental applications that require a modeling approach to handle these complex situations. To solve more practical problems such as nationwide epidemics, these applications need a large number of agents to simulate. Although parallel and distributed simulation frameworks have slowly started to address these computational needs, non-computing scientists are still hesitant to use these parallel and distributed frameworks. This paper strives to identify reasons behind these hesitations through programmability and performance analysis. ABM applications were surveyed and modeled as seven benchmark tests and implemented on UW Bothell's MASS library and RepastHPC, Oakridge National Library's RepastHPC.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thank the chair of my capstone committee, Professor Fukuda, for helping me throughout the capstone process. This project would not have been possible without his guidance, assistance, and hard work.

I would also like to thank the other members of my project committee, Prof. Stiber and Prof. Parsons, for their meticulous work in reviewing my progress reports.

Chapter 1
INTRODUCTION

To assist in the growing need to solve large scale simulations, parallel computing frameworks that would increase both computation speed and space are required. Yet non-computing users are still using simulation platforms for sequential execution.

If the tools are too cumbersome or difficult to use, then scientists will shy away from using them. It is desirable that these evaluations can shed light into what can be done to improve usability so that more would be willing to use the most up to date tools.

1.1 Motivation

ABM views simulation as an emergent collective group behavior of many agents, each with individual autonomy of computation and mobility over a simulation space. It is powerful to serve on these applications that cannot always be modeled with formulaic equations. Social, Behavioral, and Economic Sciences (SBE) is this type of application domain that observes human behavior and how humans and social, behavior, and economic forces affect each other. However, a lot of the current ABM platforms are not able to handle the amount of large scale information that is needed to perform the desired SBE simulations. As a result, parallel and distributed ABM platforms were researched. These platforms include RepastHPC and FLAME, but both platforms struggle with programmability. These struggles led to the hesitation of use from non-computing scientists. UW Bothell's MASS library [5] was developed to resolve these issues.

Since FLAME doesn't support agent mobility, it was decided to perform our evaluations on the two most available parallel and distributed ABM platforms: RepastHPC [7] and UW

Bothell's MASS library [5]. The former was developed by Oakridge National Library and the latter being our own library. Performing evaluation will shed light to advantages and disadvantages of MASS compared to RepastHPC. Analysis of these reasons will be conducted through the viewpoints of programmability and performance.

1.2 Problem Definition and Goal

This project is to expand on our C++ based sequential benchmarks [42] by developing parallel benchmark applications for each category under RepastHPC and MASS so that programmability and performance can be analyzed and compared between the two frameworks.

Chapter 2
BACKGROUND

To predict human behavior and social economic trends, comes the study of SBE. With this area of expertise receiving greater attention in the observation of group behavior comes a necessity to dive deeper into the study of these sciences. Benchmark tests will be used to provide analysis for various areas of study within SBE. Before these tests are covered, the following sections will give background on SBE and then proceed to cover what ABM is. Finally, simulators that non-computing scientists use will be discussed in the final section.

2.1 Social Behavioral and Economic Sciences (SBE)

SBE delves into human behavior and how social, behavioral, and economic forces affect the lives of humans as well as how humans in turn affect these forces. These concepts can span from low level ideas such as living cells to human society. It can also range from neurons within a brain grid to neighborhoods where people live as well as spanning to cover ideas such as space and time. [20]

These ideals are important and can impact areas such as discovering where we as humans came from, to stabilizing and improving national defense, and to protecting humans from cyber crimes. Through these ideas and principles, greater knowledge of human patterns from an individual stand point to society's stand point can be gained.

2.2 Agent Based Modeling (ABM)

The goal of studying SBE has led to the use of agent based modeling. Since SBE has models that are complex and benefit from an object oriented approach, scientists have gone towards the way of ABM which is found to be tightly coupled with SBE. ABM is an approach to

modeling that utilizes autonomous, interacting agents that can be used to cover applications such as agent behavior in the stock market, consumer markets, all the way to predicting epidemics [13] such as tuberculosis. It can also assist businesses in decision-making as well as help researchers study diseases and brain functions. By using ABM, one is able to compute large-scale simulations that would not have been possible.

2.3 Simulators

Having gained the knowledge that non-computing scientists use ABM platforms to simulate ABM applications, research was done on which platforms were used. NetLogo [41], MASON [4], Repast, and Ascape [1] are examples of platforms that are currently used that stress programmability which includes extending the simulation space from 2D to 3D as well as allowing inter-agent communication.

Although these mentioned frameworks are available to ABM scientists, there have been growing issues with scaling up their model size. These issues have resulted in scientists looking at parallel computing frameworks that would increase both computation speed and space. Issues arose from the mentioned frameworks since parallelization is normally limited to multi-threading thats built in their language.

As such, the following are several parallel and distributed ABM systems that address computational needs in both speed and space over a cluster of computing nodes. RepastHPC [7] 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. It is focused on agent migration over distributed 2D/3D space and provides agents with a view of remote computing nodes' boundary space known as ghost space. Ghost spaces, which are adjacent boundaries of remote spaces that are visible to local agents, prevents agents from moving to where other agents reside; however unless agents can read each other's logic, or their migration is serialized, parallel simulation cannot perfectly avoid agent collision at the same spot.

FLAME [23] is another parallel platform which is written in C. However, for object-oriented

programming purposes, agents and environment variables are declared in XML, similar to C++ header files. 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. FLAME is considered as a collection of communicating, state-transiting agents statically mapped over MPI ranks. From this viewpoint, FLAME would burden ABM designers with application-level manipulation of agents that need to migrate over a space.

Other conventional parallel multi-agent systems include PDES-MAS [29], MCE3J [22] as well as mobile-agent systems such as D'Agents [2], JADE [3], and Pandora [6]. However, these are coarse-grained cognitive agents to assist human decisions which make them not suitable for mega-scale fine-grained agents.

Noticing that the widely used parallel and distributed frameworks all have problems created motivation to examine common issues in parallelizing ABMs to improve user-level quality of life in regards to programmability and performance. By performing evaluation using RepastHPC and MASS, this project focuses on shedding light to advantages and disadvantages of MASS compared to RepastHPC by developing benchmark tests.

Chapter 3
BENCHMARK TEST SET DESIGN

ABM applications were sampled and then organized into seven benchmark tests that are able to encompass all surveyed domains. Section 3.1 covers design strategy in which the domains are overviewed, followed by section 3.2, the benchmark test section which elaborates on how the seven benchmark tests were formed from the ABM applications. Lastly, sections 3.3 and 3.4 provide explanations on all benchmark tests are given in detail.

3.1 Design Strategy

The following are ABM applications in each scientific discipline to give a generalized sampling of all areas within ABM.

- **Social Science**

    1. *Social Network Modeling*: simulates activities among on-line communities [8, 18]. This can be modeled as a network of agents, each exchanging information along links with its neighbors

- **Behavioral Science**

    1. *Flows*: observes the dynamics of indoor and outdoor pedestrian crowd and metropolitan emergency evacuation [30,32]. These simulations use cellular automata or observe agent movement over a 2D grid.
    2. *Organizations*: forecasts 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 [28].

- **Economic Science**

    1. *Markets*: includes bank bail-in/out [27] and Internet service provider (ISP) markets [10], 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.
    2. *Diffusions*: quite close to social network modeling where a product adoption is modeled as a word-of-mouth diffusion on social networks [10].

- **Biological/Ecological Sciences**

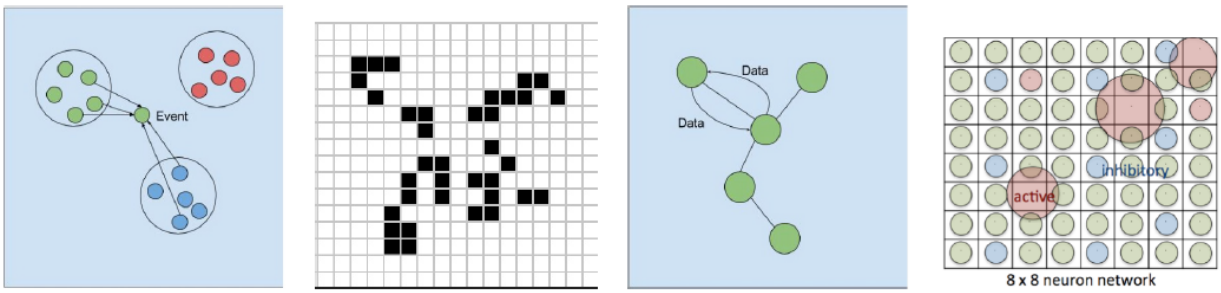
    1. *Cell-level simulation*: simulates the growth of synapses [26], identifies control mechanism of granuloma formation by TB bacteria, macrophages and T-cells [37], and cell-level blood flow throughout the microvessel network [40].
    2. *Ecosystems*: simulates an emergent collective group behavior of (artificial) lives. Examples include Wa-Tor: an ecological war between sharks and fish [16] and Sugarscape: an artificial society [17] which are both based on agent migration over a 2D space. Conways Game of Life [21], a well-known cellular automata game could also be considered as an artificial life.
    3. *Epidemic simulation*: predicts a pandemic of a given disease such as dengue fever [25] and influenza [13]. They move agents from one community to another.

- **Urban Planning**

    1. *Transport simulation*: predicts traffic flows under a given condition. The major simulators are TRANSIMS [9] and MATSim, each based on cellular automata and a queuing network, respectively.
    2. *Land use simulation*: simulates agent behaviors over households, businesses, and land areas. The model is generally combined with transport simulation.

3.2 Benchmark Tests

Upon looking at the sampling of all areas within ABM, it was found that the above applications could be categorized into seven separate benchmark tests from the viewpoint of agent mobility that reside in two categories: Static and dynamic agents.



(a) Clustered agents (b) Cellular automata (c) Network of agents (d) Dynamic communication

Figure 3.1: Static Agents

1. **Static Agents**: have no mobility over a simulation space

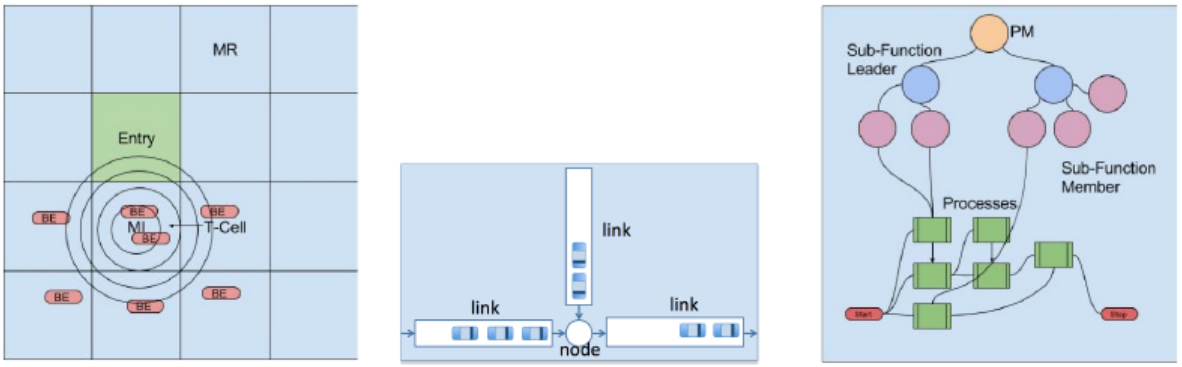
   (a) *Clustered agents*: forms one or more groups, each exchanging events with each other and sharing them among its internal agents (Figure 3.1-(a)). Market simulation (financial modeling) can be categorized into this model.

   (b) *Cellular Automata*: considers each cell as a static agent that communicates with its neighbors (Figure 3.1-(b)). This model can be simulated using Conway's Game of Life.

   (c) *Network of Agents*: is a more generalized form of cellular automata by replacing a 2D/3D grid with a network (Figure 3.1-(c)). Social network modeling and economic diffusions belong to this category.

   (d) *Agents with Dynamic Communication*: stays static but extends its communication paths to further agents as seen in the growth of synapses (Figure 3.1-(d)). Brain

Grid falls under this category.



(a) Agent migration over grid (b) Agent migration over network (c) Group creation over multi phases

Figure 3.2: Dynamic Agents

1. **Dynamic Agents**: have navigational autonomy over a space or dynamic creation over a simulation period

   (a) *Agent Migration Over Grid*: moves agents over a 2D or a 3D space as shown in Figure 3.2-(a). Most ABM applications in biology and ecology can be categorized in this model (TB).

   (b) *Agent Migration Over Network*: walks or drives agents over a network (MatSim) as illustrated in (Figure 3.2-(b)).

   (c) *Group Creation Over Multi Phases*: is a very special case in agent creation that corresponds to VDT. Figure 3.2-(c) models a developer team's transition from one to another product phase.

The goal is to study these seven categories to identify and address the current programming difficulties in parallel/distributed ABM platforms. Details on the seven benchmark tests are given in the following sections.

3.3 Static Agents

- **Financial Modeling** The Financial Modeling benchmark is a test that covers agents with static communication. The program's abstract idea is static, with the implementation being dynamic having a message passing agent. The benchmark test has clusters of agents, each agent being a bank or a firm. Our financial modeling benchmark test is an extremely simplified model that demonstrates how a bank and a firm may communicate to borrow and lend money.

  Each place may contain agents and each agent can either be a firm or a bank. A bank is an agent that will lend money to any firms that need money. A firm is an entity that may need money at which point will ask any bank available to lend it money.

  Financial Modeling functions in which all banks will first attempt to lend money to the firms in its respective cluster. If the banks run out of money or no longer has any firms to lend money to, the banks will then migrate to another cluster and check if there are any firms that need money lent to them. The number of migrations can be increased by the user to specify how many migrations the banks should do.

- **Game of Life**

  Conway's Game of Life benchmark is a test that covers cellular automata. The program has an orthogonal grid of square cells where each is categorized as dead or alive.

  In this program initial setup is done using the rand() function which generates either a 1 (alive) or 0 (dead). Each cell will abide by the following rules:

  1. Any live cell with fewer than two live neighbors dies
  2. Any live cell with two or three live neighbors lives
  3. Any live cell with more than three live neighbors dies
  4. Any dead cell with exactly three live neighbors becomes a live cell.

The above 4 statements can be summarized by the following algorithm:

1. If the sum of all nine cells is 3, the current cell's next generation will be life.
2. If the sum of all nine cells is 4, the current cell's next generation will be its current health status of life or dead.
3. Every other sum, the current cell's next generation will be death.

- **Social Network**

  The Social Network benchmark is a test that covers a network of agents. The program has an orthogonal grid of square cells where each is a person with a random set of neighbors.

  Each person has a vector of its own friends. Degrees of separation is then calculated by traversing through each neighbor's vectors up to the number of degrees of freedom. An assumption is made that there is at least 1 degree of separation.

  A vector is used to store all friends up to X degrees of freedom away. The for loop appends each friend found at each degree of freedom up to the degrees of freedom specified in main. A counter is incremented so that when the degrees of freedom specified is reached, a for loop is traversed from count up to the end of the vector to retrieve all friends within the user specified degrees of freedom.

- **Brain grid**

  The Brain Grid benchmark is a test that covers agents with dynamic communication. The program has an orthogonal grid of square cells where each is either an active, inactive, or neutral neuron. Our Brain Grid benchmark test is an extremely simplified model to see how each cell can grow its communication links to the others, based on the idea of brain theory and neural networks [39].

Active neurons grow in all 8 directions (Moore neighborhoods), neutral neurons grow in the direction it was grown into, and inactive neurons don't grow, but prohibit growth. As these connections grow, connections are made between the various neurons to create a network that loosely simulates a neural network.

3.4 Dynamic Agents

- **Tuberculosis**

  The Tuberculosis benchmark is a test that is a simplified model of tuberculosis infection that covers agents with dynamic communication. The program has an orthogonal grid of square cells where each cell is considered either infected bacteria or clean bacteria. Agents will come through and have a chance to be an infecting agent or cleaning agent.

  Agents and bacteria infect or clean radially to its 8 surrounding neighbors. Agents will also eventually run out of energy and die out after a certain number of moves.

  Infected bacteria will infect radially to all 8 neighbors while agents can either infect or clean in a similar fashion. Agents can either be cleaning or infectious, with both functions performing at double the rate of bacterias infectious spread. If an agent comes to a bacteria, it will clean radially to all 8 neighbors or spread to all 8 neighbors, depending on what type of agent it is. Agents will also move randomly between the bacteria.

- **MatSim**

  The MatSim benchmark is a test that covers agents with static communication. The program has an orthogonal grid of square cells where each cell is considered an intersection on a grid of streets and each agent is a vehicle. Our MatSim benchmark test is an extremely simplified model of traffic simulation.

  Each cell contains agents, which represents a vehicle crossing an intersection. An agent is a vehicle that will traverse the grid of intersections.

MatSim's agents utilizes Dijkstras algorithm to find the shortest paths between the different cells. Each vehicle has a randomized start point and a randomized end point. At this point, Dijkstras algorithm will be run on each specific agent where their shortest path between the two points is found. Traversal of all agents will then occur and efficiency is measured.

- **VDT**

  The VDT benchmark is a test that is a simplified model that covers agents with dynamic communication. The program has an orthogonal grid of square cells where each cell is considered a project. Agents are placed within each cell where they perform work on the project.

  There are three types of people: Project Managers, Sub-Function Leaders, and Sub-Function Members. Each Sub-Function member has an intelligence level and a resource number. Each project will have a minimum requirement of total workers, total resources, total intelligence, and number of work cycles that is needed for the completion of the project.

  Projects within the benchmark test has a set order where later projects within the test need to wait for projects prior to them to complete before they can start up. Each project will have a random number of sub-function members populated at start. When the project is up for work, a check will be made to see if enough members are tasked to the project. If there aren't, additional members will be tasked to the project so that the requirements are met at which time the work cycles will then be performed.

Chapter 4

PARALLELIZATION

Now that the benchmark tests have been discussed in detail, the parallelization frameworks used to implement these tests will now be discussed as well. MASS and RepastHPC were chosen as the two parallel frameworks because both facilitate agent mobility and covers the execution of both static and dynamic agents. With MASS being one of the frameworks to be analyzed, RepastHPC was also chosen because it is similar to MASS with the goal of of popularizing parallelization frameworks throughout non-computing scientists.

4.1 MASS Library

The MASS library [5] is available to the public in Java and C++. For the scope of these benchmark tests, the C++ version was used. MASS has two key classes: Places and Agents. Places is a multi-dimensional array of elements that can be allocated over a cluster of multi-core computing nodes. Each place is pointed to by a set of network-independent array indices and is capable of making a variety of function calls. The following are the major Place functions.

1. **Places Class**

   (a) **public Places(int handle, String className, int boundary, void *argument, int argSize int dim, int size[])** instantiates a shared array with size and a ghost space with boundary from className as passing an argument to the className constructor.

   (b) **public void* callAll(int functionId, void *arguments, int argSize)** calls the method specified with functionId of all elements as passing arguments[i] to

element[i], and receives a return value into (void *)[i].

(c) **public void exchangeAll(int handle, int function-Id, Vector <int*>*destinations)** calls from each element to a given method of all destination elements, each indexed with a vector element, and exchanges data among the elements.

(d) **public void exchangeBoundary( )** exchanges boundary data with neighboring nodes as a ghost space.

2. **Place Class**

(a) **private vector<int>size, index** maintains the size of the shared array that each element belongs to as well as the index of each array element.

(b) **public Object callMethod(int functionId, void *arguments)** is invoked from Places.callAll() and exchangeAll() so as to call a function specified with its corresponding functionId.

Agents are a set of execution instances that can reside and move from one place to another. They can also indirectly interact with other agents through variables local to the current place they are residing on. The following are the major Agent functions.

1. **Agents Class**

(a) **public Agents(int handle, String className, void *arguments, int argSize, Places *places, int population)** instantiates agents from className, passes arguments to their constructor, and populates them over a given Places, based on Agent.map().

(b) **public void* callAll(int functionId, void *arguments, int argSize, int retSize)** This is the same as Places.callAll().

(c) **public void manageAll()** updates each agent's status, based on its latest calls of migrate(), spawn(), and kill(). These methods are invoked within callAll().

2. **Agent Class**

   (a) **migrate(int[] index...)** allows a calling Agent to migrate or propagate itself to one or more Places specified with index upon Agents.manageAll().

   (b) **spawn(int nChildren, Object arguments)** spawns children as passing arguments to them.

   (c) **kill()** terminates a calling Agent.

   (d) **public Object callMethod(int functionId, Object argument)** is the same as Place.callMethod().

   (e) **public int map(int maxAgents, vector<int>size, vector<int>index)** returns the of agents to be populated at the place specified with index of a shared array with size. Note: maxAgents indicates total of agents to be populated over the array. It can be overloaded by a user.

Parallelization with MASS is done through a set of multi-threaded communicating processes that are forked over a cluster of multi-core computing nodes with libssh2 in C++ which are connected to each other through TCP sockets. Multi-threads control parallel method invocation and information exchange between Places and Agents.

4.2 RepastHPC

RepastHPC is an agent-based modeling and simulation framework that is implemented in C++ utilizing MPI to provide high-performance parallel and distributed simulations. There are Agents and Contexts in RepastHPC with agents being implemented as C++ classes, each with a unique **repast::AgentId** as well as maintaining its state represented by the variables in the corresponding classes with its behavior being described by the functions in those classes. **Contexts** are used to all the population of agents. When an agent is created, it is added to a Context with **addAgent()**. **removeAgent()** will remove the agent from the Context. Contexts can also move agents over a shared space

using **move()** and **balance()**. RepastHPC also describes simulation spaces using what they call Projections. Some projections include **SharedNetwork, SharedDiscreteSpace, SharedContinuousSpace, DiffusionLayerND,** and **ValueLayerND**. Each of these model a different type of simulation space: a 2D/3D shared discrete space, a shared contiguous space, an N-dimensional layer of diffusing values over a space, and an N-dimensional array of values accessed by agents. These spaces also provide agents with Moore2DGridQuery and RepastEdgeContentManager classes to allow them to find neighboring agents. RepastHPC also 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.

Chapter 5

ANALYSIS

Upon the completion of the implementation of the seven benchmark tests, programmability and performance analysis have been done on the tests that were coded in MASS C++ and RepastHPC. Both programmability and performance sections are broken up into the seven benchmark tests, each subcategory comparing the differences between MASS and RepastHPC.

5.1 Programmability

Programmability comparisons between MASS C++ and RepastHPC are discussed from both the quantitative and qualitative viewpoints. For the quantitative discussions, we will use Figure 5.1 to summarize lines of code (LoC) in MASS C++ and RepastHPC.

	Financial Modeling		Life		Brain Grid		Social Network		MatSim		Tuberculosis		VDT	
Code	MASS	Repast	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	311	424
Agents	227	194	223	131	467	240	332	161	185	307	143	200	335	283
Total	529	594	330	517	587	626	446	575	529	705	567	607	646	707
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	0.91	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	1.18	1.00

Figure 5.1: Lines of code in MASS C++ and RepastHPC

5.1.1 Financial Modeling

MASS C++'s implementation of Financial Modeling utilizes its Places class as its way of simulating a cluster. The Agents class is used to represent the two different types of

entities: Banks and Firms which reside in the clusters. Because there is no agent-to-agent communication within MASS, the Place that respective agents reside on is used as a way to query each agent's money and operations are then performed on each agent.

With RepastHPC's implementation of this benchmark test, agents are used as the way to simulate both firm and bank entities. Spatial projection is then used as a way to simulate clusters within this benchmark test. Although RepastHPC's spatial projection only supports passive space for agents to store, retrieve, and share data, the ability of RepastHPC to have agent-to-agent communication mitigates the issue and allows firms and banks to communicate with each other in each of the clusters.

The benchmark tests for Financial Modeling results in fewer total lines of code for MASS with RepastHPC having fewer lines of code for agent logic. This supports RepastHPC's disadvantage of requiring additional lines for initial setup. However, with MASS being able to utilize communication through its Place class and RepastHPC being able to utilize agent-to-agent communication, the difference between MASS and RepastHPC's agent code is less pronounced in this case.

5.1.2 Game Of Life

MASS C++'s Game of Life implementation utilizes its Places class as the way of simulating each life in this benchmark test. With Places being able to behave as a static collection of elements that can perform computation, using Place objects as the way to simulate each life was the most logical choice. Neighboring Places' information were exchanged as its way to perform the checks needed for this test.

RepastHPC's implementation of Game of Life utilizes agents dispersed across a spatial projection as its way of simulating each life in this benchmark test. With RepastHPC only supporting passive space for agents to store, retrieve, and share data, agents were used as the way to simulate each life's interactions with one another. Using RepastHPC's Moore2DGridQuery, checks for each life's state was able to be completed.

Looking at the lines of code for both MASS and RepastHPC's implementation of Game

of Life, one can see MASS has less total lines of code written. However, RepastHPC has fewer lines of agent code written. Figure 5.1 shows the lines of code (LoC) breakdown.

5.1.3 Social Network

The Places class is used once again as its way of representing each person in MASS' implementation of this benchmark test. Because of the lack of agent-to-agent communication under MASS, it was most logical to write this benchmark test utilizing MASS' Places class. Each place object had every other place object within the benchmark test as a neighbor so that it would be possible to traverse each other person's list of friends. RepastHPC's implementation of Social Network utilizes agents to simulate each person in this benchmark test. Because of its ability to have direct agent-to-agent communication, there is no need to have a spatial projection within this benchmark test.

MASS' implementation of Social Network has fewer total LoC, but, it has a significant amount more when it comes to agent LoC. With Social Network, RepastHPC's ability for agent-to-agent communication resulted in the logic for Social Network being able to be written in half the LoC compared to MASS.

5.1.4 Braingrid

MASS C++'s implementation of Brain Grid utilizes its Places class as its way of simulating the three different neurons in this benchmark test. Since MASS cannot create a network of agents, direct communication is not supported. That coupled with the fact that Places in MASS are able to behave as a static collection of elements that can actively perform computation, led us to using Place objects as the way to simulate Brain Grid's neuron interactions. Each of the eight surrounding neighboring Place's information were exchanged as its way to perform the checks needed for this test under MASS.

RepastHPC's implementation of Brain Grid utilizes agents dispersed across a spatial projection as its way of simulating the three different neurons. Since RepastHPC supports only passive space for agents to store, retrieve, and share data, agents were used as the way

to simulate Brain Grid's neuron interactions. RepastHPC also allows the user to query the space around an agent by utilizing its Moore2DGridQuery which exactly satisfies what is needed to be done under our Brain Grid benchmark test. As such, that was utilized to perform the spatial checks needed for this test.

Programmability for the logic of MASS and RepastHPC using lines of code as a quantitative measurement shows that although MASS has a total of less lines of code written, RepastHPC actually has fewer lines of code written for its agent portion. This is because RepastHPC requires substantially more lines for the setup of MPI-parallel and message packing/unpacking code. This can be seen on Figure 5.1. Another programmability feature for both MASS and RepastHPC in this benchmark test was the decision to simulate growing communication channels of each place/agent in MASS and agent-to-agent in RepastHPC by just relaying data from one place/agent to another place/agent for MASS and RepastHPC respectively. The reason for this programmability decision is because I felt it would more closely represent our design model of how a brain grid functions.

5.1.5 Tuberculosis

Tuberculosis' MASS C++ implementation uses its Places class as its way of simulating bacteria in this benchmark test. The Agents class is then used to simulate the two different types of agents: cleaning and infecting. With the Place class being the main bacteria, MASS' ability to have Places perform computation is utilized to determine the status of each bacteria in the benchmark test.

Because RepastHPC's spatial projection supports only passive space for agents to store, retrieve, and share data, three types of agents are utilized to simulate the Tuberculosis benchmark test in RepastHPC: bacteria, cleaning, and infecting agents. RepastHPC then utilizes its agent-to-agent communication ability as well as the built in Moore2DGridQuery to perform its necessary checks in this benchmark test.

Being that there is no need for direct agent-to-agent communication in Tuberculosis, and MASS' ability to have Places perform computation and RepastHPC's spatial projection's

inability, this benchmark test resulted in less total lines of code as well as fewer agent lines of code in MASS compared to RepastHPC. Although there are fewer lines of code in RepastHPC for environment/space, these numbers still show that there is a higher number of required lines for setup in RepastHPC. However, since logic for the benchmark test could be shifted into Places in MASS, this allowed MASS to have fewer lines of code compared to RepastHPC in its agent code.

5.1.6 MatSim

MatSim's MASS C++ implementation utilizes its Places class as its way of representing the benchmark test's version of streets and its Agents class as its way of representing the vehicles that traverse the streets. Because MASS' Places are able to perform computation, the shortest path algorithm used was able to be computed on through the Place class. RepastHPC's implementation of MatSim utilizes agents to simulate its vehicles and its spatial projection as the way to simulate the streets traversed by the vehicles. With RepastHPC only supporting passive space for agents to store, retrieve and share data, the shortest path algorithm was computed on the agents themselves.

The lines of code required to write MatSim with MASS and RepastHPC show that MASS required fewer total lines of code as well as fewer lines of code for its agent logic. Furthermore, MASS has the advantage of being able to have the shortest path algorithm computed through its Places class as well as being able to mimic a general road network with places, each maintaining an adjacency list. Because RepastHPC's spatial projection can't do this, it relies on each agent to obtain an overall adjacency matrix in the beginning before moving through its roads.

5.1.7 VDT

VDT's MASS C++ implementation uses its Places class as its way of simulating each project. The agents class is then used to simulate the leader and worker agents. With the Place class being able to perform calculation, each project's requirements are performed on each place

with each leader and worker agent performing the required work within their respective places.

With RepastHPC's implementation of VDT, three types of agents were utilized to simulate the VDT benchmark test: project, leader, and worker agents. RepastHPC then utilizes its agent-to-agent communication ability to perform its necessary checks in this benchmark test. Since this benchmark test requires the ability to create additional worker agents if the project requires more resources, The max amount of agents that may be needed for all projects to be performed is calculated prior to the execution of the test and created at startup.

In regards to this benchmark test, MASS has the advantage of being able to spawn agents as needed whereas RepastHPC has the disadvantage of needing to provide all total number of agents at the beginning.

Features	Appl.	MASS	RepastHPC
Creation Arrays	Life, TB, VDT Brain, Bank	+ Yes, with Places	+ Yes, with SharedDiscreteSpace
Graphs	SocialNet MatSim	- Mimicked by Places, using neighbors in exchangeAll()	+ Yes, with successors in the agentNetwork class
Addressing Space	Life, TB, VDT Brain, Bank	- Only array indicies 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 moore2Dspace()
Over graph	SocialNet	- Mimicked by neighbors in exchangeAll()	+ yes, with agentNetwork and DiffusionLayerND.diffusion()
Space & agents	Transport, VDT 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

Figure 5.2: Spatial Management

Features	Appl.	MASS	RepastHPC
Creation			
Population	All	- Individually with Agent.map	+ Entirely with Context.addAgent()
Multiclasses	Transport TB, VDT	+ yes, with new Places()	- One class of agents per context but ValueLayerND shared among multiple contexts
Runtime	TB, VDT	+ 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, VDT	+ 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		
Autonomy	TB	+ 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

Figure 5.3: Agent Management

Features	MASS	RepastHPC
Scenario control	– main() describes an entire scenario.	+ Context registers discrete events.
Parallelization	+ Multi-processes spawned by libssh2 + Multithreading supported + System-supported agent/data transfer – Synchronization required for agents to access the same place	+ Multi-processes spawned by MPI – No multithreading for agent manipulation within each Context – User-operated data pack/unpack + 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

Figure 5.4: Entire Simulation Management

5.1.8 Overall Programmability Comparison

Having gone through all seven benchmark tests, advantages and disadvantages for agent and spatial management under MASS and RepastHPC can be compared for all seven tests. In

general, it can be said that in regards to spatial management, RepastHPC provides both multi-dimensional spaces and any graph topologies as a simulation space, whereas MASS must manage to emulate graphs using its Places array structure. However, RepastHPC only supports passive space for agents to store, retrieve and share data when Places in MASS can behave as a static collection of elements that can actively perform computation. Figure 5.2 shows these pros and cons.

For agent management, RepastHPC requires the user to facilitate run-time agent creation and termination while being able to populate agents over an entire space from a bird's eye view. MASS allows different classes of agents to behave on the same Places, whereas RepastHPC allows different Contexts to share the same ValueLayerND space. Since MASS can't create a network of agents, direct communication is not supported, where RepastHPC does support this. Figure 5.3 shows these pros and cons.

To compare MASS and RepastHPC as a whole, MASS focuses on hiding the underlying parallel platforms from ABM applications much more than facilitating user-friendly event scheduling and data collection. RepastHPC exposes users to the underlying parallel platforms. As such, the user must understand MPI and how to handle agent/data packing, transfer, and unpacking. The user must also maintain agent lists in Context and examine agents, which makes it difficult to parallelize. Figure 5.4 shows these pros and cons.

5.2 Performance

Along with programmability evaluation, performance evaluation of the seven benchmark tests for MASS and RepastHPC were also conducted 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 5.5 and 5.6 shows the results for the seven test cases. Their spatial sizes were adjusted between 100 x 100 and 1000 x 1000 and the number of agents were adjusted to between 200 and 162000. This was so that all tests would complete below 35 seconds for their single node execution. Discussion of performance numbers is broken up into two sections, static agents for the four static benchmark tests and

dynamic agents for the three dynamic benchmark tests.

5.2.1 Static Agents

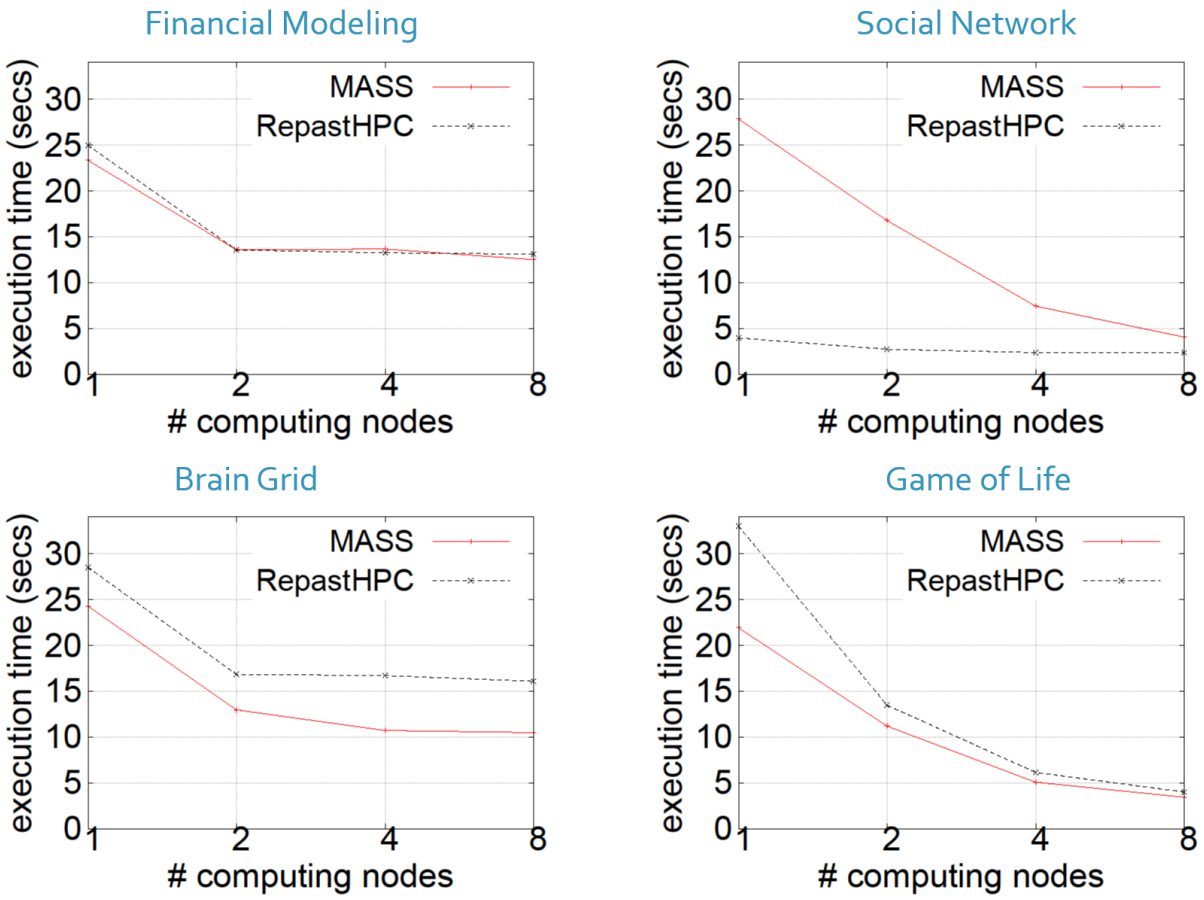


Figure 5.5: ABM Execution Performance (Static Agents)

The execution performance of Brain Grid shows that MASS outperformed RepastHPC under all computing nodes. However, under both RepastHPC and MASS, there was negligible performance increase with 4+ nodes. This is because its computation completed quickly with the current random initialization of neurons. For future benchmark testing, we need to test various initializations of neurons and to observe their effect onto performance measures.

Game of Life on the other hand showed a pretty linear performance increase as the number of nodes increased. This was due to more space made available with multiple nodes. Social Network had much faster execution under RepastHPC compared to MASS due to RepastHPC's agentNetwork class that supports a graph of static agents (agent-to-agent communication). Lastly, Financial Modeling gathered agents in groups and therefore showed less spatial parallelism. As such, the increase between 2-8 nodes were more negligible. For better performance, agents must be handled with multi-threading.

5.2.2 Dynamic Agents

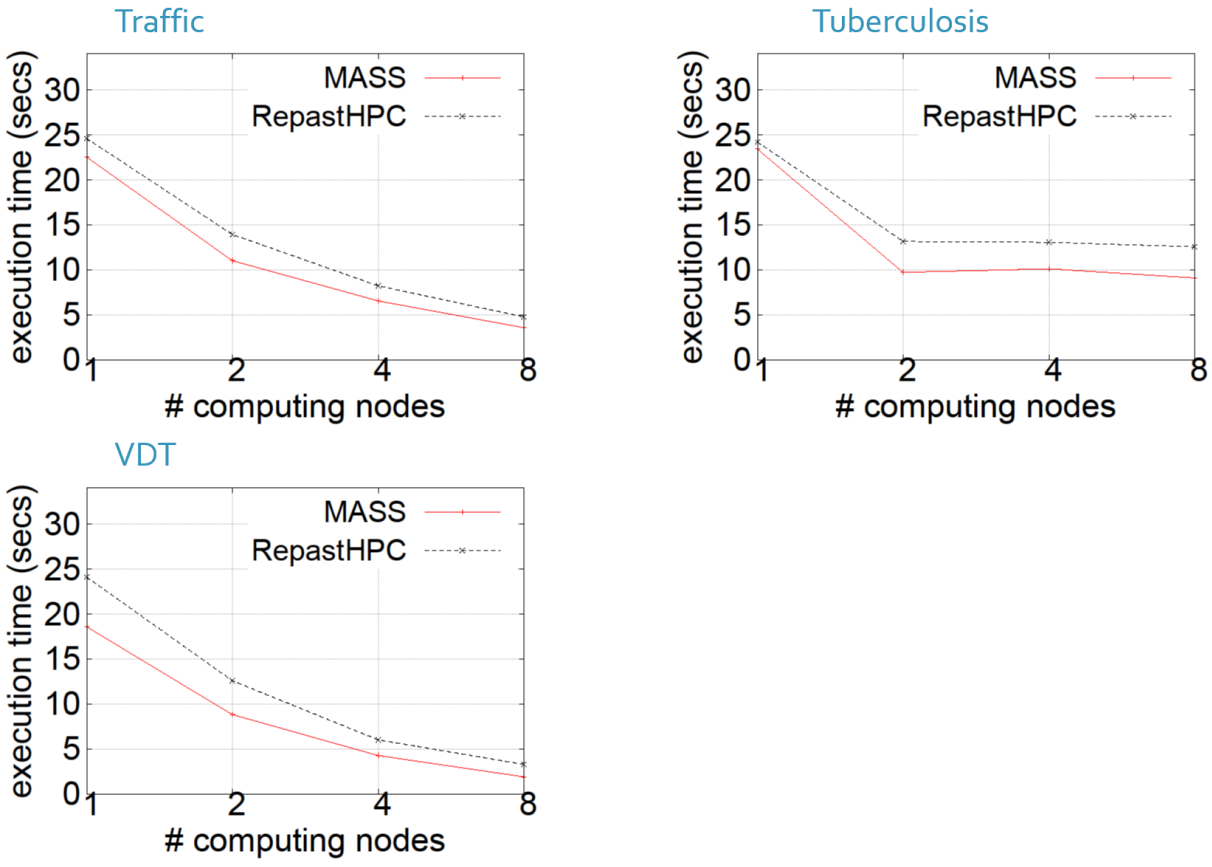


Figure 5.6: ABM Execution Performance (Dynamic Agents)

MatSim showed its scalability with its linear increase from 1 to 8. This is because the current implementation randomly populates agents whose destinations to move are also randomly chosen, so that the number of agents per each computing node is well balanced. For more realistic benchmark tests, we will generate a more restricted road network and have more vehicle agents drive to an identical destination, which simulates traffic congestions. The performance of Tuburculosis showed a slowdown from 2 to 8 nodes. This resulted from unbalanced load over 4+ computing nodes as a result from agents tending to come together. This indicates the need of dynamic load balancing. With VDT, under RepastHPC, the max amount of agents that may be needed for all projects to be performed is calculated prior to the execution of the test and created at startup. Whereas with MASS, it is dynamically created when needed. Since the benchmark test is fairly accurate in the number of total agents needed, performance between MASS and RepastHPC did not differ much because of total agent creation numbers. With the way the benchmark tests are designed with its resource, intelligence, and number of workers needed for each project being fairly consistent, an estimate of total number of agents is able to be made accurately. However, if resource, intelligence, and number of workers were to be completely randomized, then there is a chance that the total number of agents needed can have an inaccurate estimate. Furthermore, if all three categories were to be completely randomized, additional agents would need to be created on RepastHPC to account for the inaccurate estimation, which would result in a decrease in performance. One thing to note is that performance results did exhibit super-linear performance increases. This is a result of the creation of many agents. As nodes increase, more memory is made available so that the memory bottleneck would be negated in larger scale simulations. For future benchmark testing, we need to add some restrictions on allocation, de-allocation, and move of resource, intelligence, and works, which will simulate more practical development phase transitions and observe more viable performance measures.

5.2.3 Overall Performance Comparison

Looking at the seven benchmark performance values, it can be seen that RepastHPC performs slower than MASS (with the exception of Social Network). This is because of its MPI-based implementation and user-level agent serialization compared to MASS directly using sockets and automates all its serialization. To summarize, even though a cluster system allows more computing resources in CPUs and memory which allows simulation space and agents to scale, parallel simulators need to balance initial and dynamic distribution of space and agents over cluster nodes which should utilize CPU cores more efficiently.

Chapter 6
FUTURE WORK

The benchmark tests evaluated for programmability and performance under MASS and RepastHPC can be considered just the first steps for benchmarking ABM applications utilizing parallel and distributed frameworks. We will continue to explore other applications related to SBE. Although, our research discovered and organized our research domains of social science, behavioral science, economic science, biological/ecological sciences and urban planning into seven categories, additional categories can be made for more in-depth benchmarking.

Our results can also be extended and compared with FLAME. Finally, utilizing the programmability and performance evaluation of MASS and RepastHPC, I feel that agent-to-agent communication can be implemented in MASS due to the benefits that RepastHPC had with its agent-to-agent communication capabilities. An idea for the implementation of agent-to-agent communication in MASS can be based off the idea of creating a context for all agents and to allow these agents to have communication abilities with the other agents within the same complex. By doing so, it can allow simulations such as social network to be implemented in MASS with greater ease and efficiency similar to RepastHPC's application.

Chapter 7

CONCLUSION

ABM applications in Social, Behavioral, and Ecological sciences were surveyed and categorized into seven benchmark tests from the viewpoint of agent mobility. Analysis were performed under MASS and RepastHPC for their agent and spatial descriptivity as well as performance overheads incurred by their implementation. Based on our agent descriptivity analysis, we propose that parallel ABM frameworks should hide parallel-computing constructs from non-computing users as RepastHPC setup can be cumbersome and difficult to understand. Improvement of agent mobility should also be done as they can mimic human, animal, and moving objects. Performance numbers demonstrated substantial improvements with two cluster nodes; however not always with 4+ nodes. To better scale further computation, ABM frameworks should efficiently handle agents with CPU cores and to distribute a simulation space and agents uniformly over a cluster system, thus balancing computation and communication over nodes.

Finally, it was noted that FLAME is another parallel platform and thus should have programmability and performance evaluations run on these seven benchmark tests. As said in the Future Work section, fellow student Pouria Ghadimi is already in the process of completing this so those numbers should be put in comparison to the numbers found in this evaluation.

BIBLIOGRAPHY

[1] Ascape guide. http://ascape.sourceforge.net/.

[2] D'agents: Mobile agents at dartmouth college. http://agent.cs.dartmouth.edu/.

[3] Jade site java agent development framwork. http://jade.tilab.com.

[4] Mason multiagent simulation toolkit. http://cs.gmu.edu/eclab/projects/mason/.

[5] Mass library. http://depts.washington.edu/dslab/mass/.

[6] Pandra flexible monitoring system. https://pandorafms.com/.

[7] Repast for high performance computing. https://repast.github.io/repast hpc.html.

[8] Chee Siang Ang and Panayiotis Zaphiris. Simulating Social Networks of Online Communities: Simulation as a Method for Sociability Design. In Tom Gross, Jan Gulliksen, Paula Kotz, Lars Oestreicher, Philippe Palanque, Raquel Oliveira Prates, and Marco Winckler, editors, *Human-Computer Interaction INTERACT 2009*, pages 443–456. Springer Berlin Heidelberg, 2009.

[9] C.L. Barrett. Transsims(transportation analysis simulation system. 1999.

[10] E. Bohnabeau. 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. 2002.

[11] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7280–7287, May 2002.

[12] Paul Borrill and Leigh Tesfatsion. Agent-based modeling: the right mathematics for the social sciences? *Economics Working Papers (20022016)*, June 2011.

[13] Dennis L. Chao, M. Elizabeth Halloran, Valerie J. Obenchain, and Ira M. Longini, Jr. FluTE, a Publicly Available Stochastic Influenza Epidemic Simulation Model. *PLOS Computational Biology*, 6(1):e1000656, January 2010.

[14] Ferdinando Chiacchio, Marzio Pennisi, Giulia Russo, Santo Motta, and Francesco Pappalardo. Agent-Based Modeling of the Immune System: NetLogo, a Promising Framework, 2014.

[15] Christophe Deissenberg, Sander Van Der Hoog, and Herbert Dawid. EURACE: A Massively Parallel Agent-Based Model of the European Economy, November 2008.

[16] A.K. Dewdney. Computer recreations shark and fish wage an ecological war on the torodal planet wa-tor. scientific american pp. 14-22. 1984.

[17] Joshua M. Epstein and Robert Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. Brookings Institution Press, October 1996. Google-Books-ID: xXvelSs2caQC.

[18] F. Cicirelli, A. Guerrieri, G. Spezzano, A. Vinci, O. Briante, A. Iera, and G. Ruggeri. Edge Computing and Social Internet of Things for large-scale smart environments development. *IEEE Internet of Things Journal*, pages 1–1, 2017.

[19] G. Fortino, R. Gravina, W. Russo, and C. Savaglio. Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach. *Computing in Science Engineering*, 19(5):68–76, 2017.

[20] Munehiro Fukuda. Parallel-computing support for agent-based models in social, behavioral, economic, and their related sciences. March 2017.

[21] Martin Gardner. Mathematical Games. *Scientific American*, 223(4):120–123, October 1970.

[22] Les Gasser and Kelvin Kakugawa. Mace3j: fast flexible distributed simulation of large, large-grain multi-agent systems. in proc. of the first international joint conference on autonomous agents and multiagent systems - aamas-2002, bologna, italy, november 2001. acm. 2001.

[23] Chris Geenough and Mike Holcombe. Flame flexible large-scale agent modeling environment, http://www.flame.ac.uk.

[24] Nigel Gilbert and Steven Bankes. Platforms and methods for agent-based modeling. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7197–7198, May 2002.

[25] Lus F. O. Jacintho, Andr F. M. Batista, Terry L. Ruas, Maria G. B. Marietto, and Fbio A. Silva. An Agent-based Model for the Spread of the Dengue Fever: A Swarm Platform Simulation Approach. In *Proceedings of the 2010 Spring Simulation*

Multiconference, SpringSim '10, pages 2:1–2:8, San Diego, CA, USA, 2010. Society for Computer Simulation International.

[26] F. Kawasaki. Accelerating large-scale simulations of coortical neuronal network development. master's thesis, mscsse, university of washington bothell. 2012.

[27] Peter Klimek, Sebastian Poledna, J. Doyne Farmer, and Stefan Thurner. To bail-out or to bail-in? Answers from an agent-based model. *Journal of Economic Dynamics and Control*, 50:144–154, January 2015.

[28] R.E. Levitt. Vdt computational emulation models of organizations: State of the art and pracice. technical report, stanford university, http://web.stanford.edugroup/vdt/vdt.pdf. 2000.

[29] Brian Logan and Georgios Theodoropoulos. The distributed simulation of multi-agent systems. proceedings of the ieee, vol.89(no.2):174186. 2001.

[30] Liu C. Bhaduri B. Lu, W. 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.

[31] Takashi Masuda and Yoshifumi Masunaga. *Worldwide Computing and Its Applications: International Conference, WWCA '97, Tsukuba, Japan, March 10-11, 1997 Proceedings.* Springer Science & Business Media, July 1997. Google-Books-ID: Fko6NWqCAZsC.

[32] Allbeck J.M. Badler N.I. Nuria, P. Controlling individual agents in high-density crowd simulation. in: Proc. of the 2007 acm siggraph/eurographics symposium on computer animation. August 2017.

[33] P. M. Borst, M. Corbin, and P. S. Sapaty. WAVE processing of networks and distributed simulation. In *Proceedings of 3rd IEEE International Symposium on High Performance Distributed Computing*, pages 61–69, August 1994.

[34] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling Individual Agents in High-density Crowd Simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[35] Anne Rogers, Martin C. Carlisle, John H. Reppy, and Laurie J. Hendren. Supporting Dynamic Data Structures on Distributed-memory Machines. *ACM Trans. Program. Lang. Syst.*, 17(2):233–263, March 1995.

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

[37] Ganguli S. Kirschner D. Segovia-Juarez, J.L. Identifying control mechanisms of granuloma formation during m. tuberculosis infection using an agent-based model. journal of theoretical biology vol.231(no.3), 357. 2004.

[38] Jose L. Segovia-Juarez, Suman Ganguli, and Denise Kirschner. Identifying control mechanisms of granuloma formation during M. tuberculosis infection using an agent-based model. *Journal of Theoretical Biology*, 231(3):357–376, December 2004.

[39] M. Stiber Segundo, J.P. and J.-F. Vibert. "synaptic coding of spike trains. entrainment across synapses of one neuron by another", the handbook of brain theory and neural networks, m. arbib, ed. 1995.

[40] Bryan C Thorne. Multi-cell Agent-based Simulation of the Microvasculature to Study the Dynamics of Circulating Inflammatory Cell Trafficking | SpringerLink.

[41] Uri Wilensky. Netlogo, http://ccl.northwestern.edu/netlogo/.

[42] Caleb Yang. Caleb yang's css 499 undergraduate term report, https://depts.washington.edu/dslab/mass/reports/calebyang$_sp17.pdf$. 2018.